Manual

# JavaScript






Purificación Rives Alba






# MANUAL DE JAVASCRIPT



## Formación para el Empleo



Horas Recomendadas de Formación **40**



© Edita: EDITORIAL CEP S.L.

© Purificación Ribes Alba

C/ Dalia nº 20. Polígono El Lomo

28970 Humanes de Madrid (Madrid)

Tlf. 902 108 209

Edición: enero 2011

ISBN papel: 978-84-681-1378-4 / ISBN pdf: 978-84-681-5036-9

Depósito Legal: M-3015-2011



Imprime: PUBLICEP, S.L.

C/ Violeta nº 19. Polígono El Lomo

28970 Humanes de Madrid (Madrid). Tfl. 91 609 4176

## AUTOR

*Purificación Ribes Alba*

# Índice

## MÓDULO ÚNICO. MANUAL DE JAVASCRIPT

**TEMA 1. INTRODUCCIÓN ........................................................................11**

- ¿Qué es JavaScript?
- Los orígenes de JavaScript
- Estándares
- Formas de incluir código JavaScript
- Etiqueta noscript
- Posibilidades y limitaciones
- JavaScript en los distintos entornos

LO QUE HEMOS APRENDIDO ........................................................................21

**TEMA 2. DEFINICIONES Y CONCEPTOS BÁSICOS..............................................23**

- ¿Qué es un script?
- Las sentencias
- Las palabras reservadas
- Documentando nuestros programas
- La sintaxis de JavaScript
- Nuestro primer script

LO QUE HEMOS APRENDIDO ........................................................................29

**TEMA 3. ELEMENTOS DE PROGRAMACIÓN BÁSICA......31**

- Las variables
- Los tipos de datos
- El alcance de las variables
- Los arrays
- INPUT y prompt
- Operadores: asignación, incremento/decremento, lógicos, aritméticos, etc

LO QUE HEMOS APRENDIDO ......43

**TEMA 4. ESTRUCTURAS DE CONTROL DE FLUJO ......45**

- Estructuras básicas de control de flujo
- Bifurcaciones
- Break y Continue
- Bucles
- Anidamiento de dichas estructuras

LO QUE HEMOS APRENDIDO ......55

**TEMA 5. FUNCIONES PREDEFINIDAS DE JAVA SCRIPT ......57**

- ¿Qué es una función?
- Funciones que manejan variables de texto
- Funciones que manejan arrays
- Funciones que manejan números

LO QUE HEMOS APRENDIDO ......67

**TEMA 6. FUNCIONES DEFINIDAS POR EL PROGRAMADOR ......69**

- Utilidad de las funciones definidas por el programador
- Creación de funciones
- Llamadas a nuestras funciones
- Valor de retorno de una función
- Variables en una función. Ámbito de las variables: globales y locales

LO QUE HEMOS APRENDIDO ......79

**TEMA 7. OTRAS ESTRUCTURAS DE CONTROL** ........................................................................ **81**

- El bucle while
- El bucle do...while
- Diferencia entre ambos
- Utilizar switch para establecer diferentes condiciones
- Anidamiento

LO QUE HEMOS APRENDIDO ........................................................................ 87

**TEMA 8. DOM** ........................................................................ **89**

- ¿Qué es DOM?
- Tipos de nodos
- El acceso a los nodos
- La manipulación de los nodos
- Manejo de los atributos de la página mediante DOM

LO QUE HEMOS APRENDIDO ........................................................................ 95

**TEMA 9. OBJETOS. EL OBJETO WINDOWS** ........................................................................ **97**

- ¿Qué es un objeto?. Propiedades y métodos
- Objetos predefinidos de JavaScript. Jerarquía de los objetos de JavaScript
- ¿Qué es el objeto window?
- Propiedades del objeto window
- Métodos del objeto window

LO QUE HEMOS APRENDIDO ........................................................................ 103

**TEMA 10. EVENTOS** ........................................................................ **105**

- Eventos
- Tipos de eventos
- Manejadores de eventos
- Tipos de manejadores de eventos

LO QUE HEMOS APRENDIDO ........................................................................ 111

**TEMA 11. EL OBJETO DOCUMENTO .................................................................................113**

- Definición del objeto documento. Propiedades y métodos del documento
- El objeto image. Propiedades de image
- El objeto anchor. Propiedades de anchor
- El objeto link. Propiedades de link

LO QUE HEMOS APRENDIDO......................................................................................................................119

**TEMA 12. OBJETOS DE LOS TIPOS DE DATOS ..........................................................................121**

- El objeto string. Propiedades y métodos de string
- El objeto number. Propiedades y métodos de number
- El objeto date. Propiedades y métodos de date
- El objeto math. Propiedades y métodos de math
- El objeto array. Propiedades y métodos de array

LO QUE HEMOS APRENDIDO ....................................................................................................................129

**TEMA 13. EL OBJETO FORMULARIO .....................................................................................131**

- Definición del objeto formulario. Propiedades y métodos del formulario
- El objeto cuadro de texto. Propiedades y métodos de los cuadros de texto
- El objeto checkbox. Propiedades y métodos de checkbox
- El objeto radiobutton. Propiedades y métodos de radiobutton
- El objeto button. Propiedades y métodos de button
- El objeto select. Propiedades y métodos de select

LO QUE HEMOS APRENDIDO ....................................................................................................................139

**TEMA 14. OTROS OBJETOS ......................................................................................................141**

- El objeto navigator. Propiedades y métodos de navigator
- El objeto history. Propiedades y métodos de history
- El objeto location. Propiedades  y métodos de location
- El objeto frame.  Propiedades y métodos de frame

LO QUE HEMOS APRENDIDO ..........................................................................................................147

# MÓDULO ÚNICO.

# Manual de Javascript

**TEMA 1: Introducción**

**TEMA 2: Definiciones y conceptos básicos**

**TEMA 3: Elementos de programación básica**

**TEMA 4: Estructuras de control de flujo**

**TEMA 5: Funciones predefinidas**

**TEMA 6: Funciones definidas por el programador**

**TEMA 7: Otras estructuras de control**

**TEMA 8: DOM**

**TEMA 9: Objetos. El objeto Windows**

**TEMA 10: Eventos**

**TEMA 11: El objeto documento**

**TEMA 12: Objetos de los tipos de datos**

**TEMA 13: El objeto formulario**

**TEMA 14: Otros objetos**

**OBJETIVOS DEL MÓDULO:**

- Aprender las peculiaridades de JavaScript y sus posibilidades como lenguaje de programación
- Conocer los conceptos y estructuras básicas de programación y su aplicación concreta a JavaScript.
- Ser capaz de manejar los objetos de JavaScript a través de los posibles eventos.
- Desarrollar aplicaciones en JavaScript para mejorar el aspecto y utilidad de nuestras páginas web.

# TEMA 1

## Introducción

- ¿Qué es JavaScript?
- Los orígenes de JavaScript
- Estándares
- Formas de incluir código JavaScript
- Etiqueta noscript
- Posibilidades y limitaciones
- JavaScript en los distintos entornos

**OBJETIVOS:**

- Saber qué es JavaScript y qué posibilidades tiene como lenguaje.
- Conocer la historia de su origen y desarrollo.
- Poder identificar las distintas maneras de integrar scripst en nuestras páginas.

## ¿QUÉ ES JAVASCRIPT?

JavaScript es un lenguaje de los denominados lenguajes de scripting. Los scripts (script se traduce como guión, literalmente) son archivos de órdenes, programas por lo general simples. Es por esto que no podemos definir JavaScript como un lenguaje de programación en un sentido estricto, pero sin embargo sí nos permite crear páginas dinámicas, con algunos efectos realmente interesantes y que mejoren considerablemente su aspecto. Nos permite tener cierta interacción con el usuario de nuestras páginas, reconocer determinados eventos que se puedan producir y responder a éstos adecuadamente. Podemos, por ejemplo, añadir elementos con movimiento que recuerdan a las animaciones Flash. Incluso podemos crear algunos programas más complejos que manejen estructuras de datos.

Tratándose de un lenguaje de script, los programas que realicemos, no necesitarán ser compilado. Los lenguajes de scripting son lenguajes interpretados. Esto en la práctica significa que cuando trabajemos con JavaScript escribiremos nuestro programa y podremos ejecutarlo de forma directa, sin necesidad de hacer nada más.[1]

Resumiendo: trabajar con JavaSript es sencillo y rápido, los resultados pueden ser muy satisfactorios y aunque el lenguaje tenga algunas limitaciones, permite al programador controlar lo que ocurre en la página.

## LOS ORÍGENES DE JAVASCRIPT

En un principio se creó HTML para permitir el manejo de los distintos elementos que pueden aparecer en una página. Con el tiempo, sin embargo, se fue haciendo insuficiente para cubrir las necesidades que iban apareciendo a medida que éstas se hacían cada vez más sofisticadas. Conforme pasaba el tiempo, se requería progresivamente más complejidad en las posibles acciones y HTML resultó insuficiente para responder a este requerimiento, ya que se trata de un lenguaje muy estático.

En los 90 se creó Java, que supuso un gran avance en esta tecnología. Java permite desarrollar pequeños programas que se incrustan en la propia página, llamados applets. Fue desarrollado por Sun Microsystems. Más tarde, Brendan Eich de Netscape, también desarrollaría LiveScript (al que llamó al comienzo Mocha), y que es el antecedente de JavaScript. Mucho más sencillo que Java, permitía desarrollar pequeños programas en las páginas. Tras la alianza de Netscape con Sun Microsystems, se diseñaría definitivamente JavaScript, al que se consideró el “hermano pequeño” de Java, más asequible para personas incluso que no conozcan demasiado de programación y sólo útil en el ámbito de las páginas web.

Netscape 2.0 fue el primer navegador capaz de interpretar JavaScript, más tarde le seguirían los de Microsoft. Hoy día, y como consecuencia de su creciente popularidad, cualquiera de los modernos navegadores tiene capacidad de interpretar los scripts de JavaScript.

Tenemos que dejar claro que aparte del origen común y cierta similitud en el nombre, Java y JavaScript son dos lenguajes diferentes. Java es un lenguaje mucho más robusto y potente, es necesario compilarlo y está orientado a objetos. Es además es un lenguaje fuertemente tipado: las variables necesariamente deben declararse de un tipo y no se puede usar con otro tipo de valores, no es así (como ya veremos) en JavaScript.

## ESTÁNDARES

Ecma (European Computer Manufacturers Association) es una organización cuyo objetivo principal es el desarrollo de estándares y reportes técnicos con la finalidad de facilitar la creación de estándares en el uso de las tecnologías de la información y telecomunicaciones, promover su uso y en lo posible, hacerlos públicos en los distintos medios. Es una organización de ámbito internacional, por ello desde 1994 pasó a denominarse Ecma Internacional.

[1] *Los lenguajes compilados necesitan de un proceso por el cual el programa que escribimos (fuente) se traduce a lenguaje máquina.*

A mediados de 1997 publicó la primera edición del estándar ECMA-262 que definió el ECMAscript. Dicho estándar fue en realidad, el adoptado por Netscape para diseñar JavaScript, Más tarde el estándar ISO/IEC-16262 incluyó algunas pequeñas modificaciones. La tercera versión, publicada en 1999, es la que utilizan los actuales navegadores y puede ser consultado de forma gratuita en la siguiente dirección: http://www.ecma-international.org/publications/standards/Ecma-262.htm. Se encuentra actualmente en desarrollo la cuarta versión de ECMA-262, que según parece podría incluir novedades como paquetes, definición explícita de clases, etc.

## FORMAS DE INCLUIR JAVASCRIPT

Si usted es usuario de windows, ya dispone de todo lo que necesita para comenzar a trabajar con JavaScript. Lo básico para comenzar es:

Un editor de textos. En un principio es válido el bloc de notas de Windows. Ahora bien, existen editores más completos y que le facilitarán la tarea de escribir el código. Eso sí, no olvide guardar el archivo con la extensión adecuada. (.htm o .js, según los casos).

Un navegador, que puede ser el que use habitualmente.

Sería aconsejable que tuviera al menos algunas nociones básicas de HTML Daremos por hecho que es así y aunque es imprescindible incluir código HTML en los ejemplos, no nos detendremos en explicarlo.

Nuestros scripts pueden integrarse de las siguientes maneras en nuestras páginas web:

Dentro del propio documento HTML:

Aunque podamos poner nuestro código JavaScript en cualquier parte del documento, es recomendable que lo incluyamos al inicio, en la cabecera del documento y dentro de la etiqueta <head>. En cualquiera de los casos deberá siempre comenzar por la etiqueta <script> y acabar con </script> para identificarlo. Además deberá llevar el atributo type, en el caso de JavaScript es siempre text/javascript.

Veamos un ejemplo sencillo y que puede probar a escribir en el bloc de notas, para que quede más claro:

Ejemplo 1.1

Ejemplo de código en el propio documento HTML

```
<html>
<head>
        <title>Ejemplo de código JavaScript en el propio documento</title>
        <script type=”text/javascript”>
            alert(“Esto es mensaje de prueba”);
        </script>
</head>
<body>
        <p>Y esto un párrafo de texto.</p>
</body>
</html>
```

Guarde este archivo con el nombre que desee, pero no olvide ponerle la extensión .htm, podemos llamarle: ejemplo1.htm. A continuación, pruebe a cargarlo en su navegador (bastará con que haga click un par de veces sobre él con el ratón o ponga en la dirección del navegador el disco en que lo grabó, por ejemplo C: y a continuación busque entre los directorios su localización). El resultado será una pequeña ventana con el mensaje: “Esto es un mensaje de prueba”, mientras que en la página normal aparecerá el mensaje de texto.

Será oportuno el uso de esta forma siempre que se trate de un bloque pequeño de código o cuando se quieran incluir instrucciones específicas en un documento HTML que completen dichas instrucciones o funciones.

En un archivo externo:

Escribiremos nuestro programa JavaScript en un archivo externo con la extensión .js y lo enlazamos en el documento HTML por medio de la etiqueta <script>. En un mismo documento HTML podemos enlazar todos los archivos que sean necesarios, especificando cada uno de ellos dentro de su correspondiente etiqueta <script>. En este caso además del atributo type deberá definirse el atributo scr indicando la URL correspondiente al archivo externo de JavaScript a enlazar.

Lo veremos más claro con un ejemplo. En este caso es el mismo programa de antes, pero lo hemos puesto en un archivo llamado “mensaje.js” y le llamamos desde el programa principal en HTML:

Ejemplo 1.2

Ejemplo de código en documento aparte. Éste sería el HTML.

```
<html>
<head>
        <title>Ejemplo de código JavaScript en otro documento</title>
        <script type=”text/javascript” src=”mensaje.js”></script>
</head>
<body>
        <p>Y esto un párrafo de texto.</p>
</body>
</html>
```

Archivo mensaje.js

```
alert(“Esto es un mensaje de prueba”);
```

En principio pudiera parecer que este método es un tanto rebuscado, que son ganas de complicarnos la vida innecesariamente. Sin embargo resulta más recomendable. Con él estamos separando JavaScript de HTML. La primera ventaja es que tenemos un código mucho más claro y fácil de reutilizar para futuras páginas. Si en algún momento tenemos necesidad de modificarlo, es más sencillo frente a las otras formas, ya que requiere sólo que vayamos a hacer dicha modificación en el archivo JavaScript e inmediatamente se refleja en todas las páginas que lo enlacen.

En los propios elementos de HTML:

Aunque sea posible, también es el más desaconsejado. Consiste en incluir instrucciones JavaScript dentro del propio código HTML.

Siguiendo con el mismo ejemplo anterior, para que lo veamos:

Ejemplo 1.3

Ejemplo de código integrado en el propio HTML.

```
<html>
<head>
        <title>Ejemplo de código JavaScript en el propio documento</title>
</head>
<body>
        <p onclick="alert('Esto es un mensaje de prueba')">Y esto es un texto.</p>

</body>
</html>
```

En este caso hay una diferencia con los anteriores, para que el mensaje aparezca, deberá hacer click sobre la página (onclick).

Como decíamos anteriormente, es el menos aconsejable pues dificulta el mantenimiento del programa. De hecho existe un paradigma en el uso de JavaScript, denominado “JavaScript no obstructivo” que aboga por que no se utilice en absoluto. Este patrón pretende hacer una clara separación entre lo que es la estructura del documento (HTML) del comportamiento programático (JavaScript, en este caso), el diseño (CSS), y distinguiendo entre las diversas capas. Incluir de esta manera JavaScript supone “ensuciar”, por así decir, el código excesivamente. En la medida de lo posible evitaremos su uso.

## ETIQUETA NOSCRIPT

Como comentamos en el primer párrafo JavaScript es un lenguaje de scripts. Este tipo de lenguajes son lenguajes interpretados. Esta interpretación se hace, al contrario que otros lenguajes como PHP que se realiza por parte del servidor, en el propio navegador del usuario. Algunos usuarios pueden haber decidido bloquear el uso de JavaScript al entender que de esta manera navegan de forma más segura. También algunos navegadores pueden incluso bloquear, no disponer de soporte de JavaScript o ser demasiado antiguos para interpretarlo.

Si nuestras páginas requieren JavaScript para su correcto funcionamiento, es aconsejable incluir un mensaje que avise al usuario indicándole esta incidencia, por si llegara a producirse. Es una forma de darle la opción de decidir si activa JavaScript para poder disfrutar de todos los servicios de nuestra página al 100% . Igualmente le podemos sugerir, caso de que tenga una versión anticuada del navegador, la posibilidad de cambiarla por otra más moderna.

A tal fin disponemos en HTML de la etiqueta <noscript>, que nos permite hacer que aparezca un aviso de lo que está ocurriendo.

Pongamos un ejemplo de cómo podría ser:

Ejemplo 1.4
Mensaje para aquellos que no visualicen correctamente JavaScript.

```
<head>
</head>
<body>
        <noscript>
        <p>Bienvenido a Mi página, visitante </p>
        <p>Requiere para su funcionamiento el uso de JavaScript.
        Si lo has deshabilitado intencionadamente, por favor vuelve a activarlo.</p>
</noscript>
</body>
```

Observe que debe incluirse dentro de las etiquetas de <body> y además suele situarse al inicio de ésta.

## POSIBILIDADES Y LIMITACIONES

Muy probablemente sean millones las páginas de Internet que en menor o mayor medida, utilicen hoy día JavaScript. Desde su aparición, obtuvo enseguida una gran acogida y fue usado de forma masiva. Más tarde sufrió un desplazamiento con la aparición de Flash, ya que éste permitía realizar algunas acciones imposibles con JavaScript. Pero, con las aplicaciones AJAX en JavaScript, volvió a recobrar de nuevo su popularidad.

Los scripts, por ejemplo, no pueden comunicarse con recursos pertenecientes al mismo dominio desde el que se descargaron. No pueden cerrar ventanas que no haya abierto ellos, y éstas tienen limitaciones en cuanto al tamaño.

Tampoco pueden acceder a los archivos del ordenador del usuario para leerlos o cambiar su contenido, ni cambiar opciones del navegador.

Algunas de las limitaciones pueden obviarse firmando digitalmente nuestros scripts, de forma que el usuario pueda dar su permiso para estas acciones.

## JAVASCRIPT EN LOS DISTINTOS ENTORNOS

Los navegadores actuales disponen de soporte para JavaScript. En el caso del IE utiliza Jscript y el resto JavaScript.

Aunque en su origen se creara para ser utilizado en el entorno web, su popularidad ha llegado a ser tal que su uso se ha extendido a otros entornos más allá de la red y no relacionados necesariamente con programación web.

Muchos programas permiten incluir código en JavaScript. Adobe permite dicha posibilidad en los archivos PDF y en herramientas como Flash y Flex, que incluyen un dialecto.

También es el caso de Photoshop. Y algunas aplicaciones lo utilizan en la programación de sus Widgets, como es el caso de Yahoo Widgets.

## LO QUE HEMOS APRENDIDO: tema 1

→ A definir JavaScript y sus posibilidades.

→ Lo que necesitamos para comenzar a trabajar con JavaScript.

→ Desarrollar pequeños scripts que muestran mensajes.

→ La historia de JavaScript y su importancia dentro de la programación web.

## ANOTACIONES

# TEMA 2

# Definiciones y Conceptos Básicos

- ¿Qué es un script?
- Las sentencias
- Las palabras reservadas
- Documentando nuestros programas
- La sintaxis de JavaScript
- Nuestro primer script

**OBJETIVOS:**

- Distinguir los conceptos más básicos de la programación en JavaScript
- Aprender la importancia de documentar nuestros programas y cómo hacerlo.
- Reconocer la manera correcta de escribir nuestros scripts.

## ¿QUÉ ES UN SCRIPT?

A estas alturas ya podemos quizás intuir muchas cosas sobre lo que son los scripts. Hemos hablado de ellos e incluso escrito alguno. Se trata de programas sencillos. Una de las peculiaridades que tienen es que se guardan en archivos de tipo texto.

En el caso que nos ocupa de la programación web, podemos distinguir dos tipos: aquellos que se interpretan del lado del servidor (como es el caso del PHP) y aquellos que lo son del lado del cliente, como es JavaScript. Para identificar este segundo tipo, los script se preceden de la etiqueta <script>.

Sin embargo los scripts no son exclusivos de los lenguajes para web, ni siquiera son especialemente novedosos, ya en el entorno DOS existían los denominados archivos por lotes o batch. Windows reconoce, además de JavaScript, otros lenguajes de scripts.

## LAS SENTENCIAS

Nos referimos con sentencias a cada una de las órdenes que un lenguaje es capaz de entender y por supuesto, llevar a cabo. De los ejemplos anteriores ya podemos ir apuntando algunas y a lo largo de esta publicación se irán estudiando y detallando muchas más.

Cada sentencia tiene una sintaxis específica, esto es, la forma en que debemos escribirla, si debe añadirse argumentos, etc.

Deducimos fácilmente que en realidad un script es una sucesión de sentencias, colocadas en un orden lógico, de forma que al ser ejecutadas lleven a cabo la acción especificada.

## LAS PALABRAS RESERVADAS

Son palabras muy especiales en cualquier lenguaje. Su significado está previamente establecido y la posición que ocupa dentro de una sentencia se determina por la sintaxis de dicha sentencia.

Encontrará una lista de las palabras reservadas de JavaSript en la tabla siguiente:

Tabla 2.1
Lista de palabras reservadas.

| | | | | | |
|---|---|---|---|---|---|
| Abstract | Boolean | Break | Byte | Case | Catch |
| Char | Class | Const | Continue | Default | Do |
| Double | Else | Extends | False | Final | Finally |
| Float | For | Function | Goto | If | Implements |
| Import | In | Instanceof | Int | Interface | Long |
| Native | New | Null | Package | Private | Protected |
| Public | Return | Short | Static | Super | Switch |
| Synchronized | This | Throw | Throws | Transient | True |
| Try | Var | Void | While | With | |

Debemos tener en cuenta que no podemos usar una palabra reservada para nombrar a una variable o una constante, ya que esto provocaría un error.

## DOCUMENTANDO NUESTROS PROGRAMAS

A medida que desarrollemos programas será imprescindible que los documentemos. Esto sencillamente consiste en hacer algunos comentarios explicando lo que hacemos en un algoritmo, qué guardamos en una variable o en qué consiste una operación a la que la sometemos. Trataremos de escribir en lenguaje humano lo que hacemos en lenguaje de programación. Puede parecer algo superfluo, una pérdida de tiempo, pero sin embargo es realmente importante. Supongamos que escribimos un programa y transcurrido un tiempo surge la necesidad de hacer alguna modificación. Si en su momento no tomamos la precaución de documentarlo, puede ser que no recordemos estas cuestiones y sólo hacer una pequeña corrección o añadir un detalle, nos lleve el mismo tiempo (o casi) que en su momento desarrollarlo. Otro caso en que tiene utilidad es cuando se trabaja en equipo y se comparte código fuente.

En el caso de JavaScript tenemos la posibilidad de hacer dos tipos de comentarios:

- De una sola línea:

Irán precedidos por dos barras, de la siguiente manera:

Ejemplo 2.1

Comentario de una sola línea

```
// El cálculo del IVA se calcula en éste caso aplicando un 18%
```

- De varias líneas:

El texto correspondiente al comentario lo pondremos entre los símbolos /* y */. Al ocupar varias líneas, son útiles en el caso de que queramos hacer comentarios más extensos. Veamos un ejemplo:

Ejemplo 2.2

Comentario que ocupa varias líneas.

```
/* La función IVA() definida a continuación, realiza un cálculo del IVA correspondiente, para ello deberán irse introduciendo los distintos importes y el código del producto. Según el producto se le aplica un IVA del 4, 8 ó 18% */
```

En ambos casos, JavaScript entiende que no se trata de una orden o sentencia a ejecutar, sino simplemente de un comentario que deberá guardar.

## LA SINTAXIS DE JAVASCRIPT

Con sintaxis nos estamos refiriendo a cómo deberán ser escritas las distintas instrucciones para que un lenguaje sea capaz de interpretarlas. Ésta está ya predefinida. En el caso de las distintas sentencias, ya iremos viendo cada caso particular. Sin embargo, queremos destacar lo siguiente:

La sintaxis de JavaScript es bastante similar a la de Java, ya que tienen orígenes comunes. También se parece en algo a C.

A diferencia de otros lenguajes, JavaScript admite que al final de cada línea se omita el punto y coma. En realidad sólo sería necesario en el caso de poner en la misma línea varias sentencias. No es habitual, en otros lenguajes es imprescindible acabar cada instrucción con punto y coma. Es recomendable habituarnos a poner el punto y coma al final de la instrucción, a pesar de no ser necesario, ya que siempre será un buen hábito de cara a aprender nuevos lenguajes.

Los bloques de instrucciones que van dentro de las estructuras de programación van encerradas entre llaves, como veremos en su momento.

Debemos respetar las mayúsculas y minúsculas, ya que JavaScript distingue entre ambas. Si a una variable le damos un nombre en minúsculas y nos equivocamos al utilizarlas luego, tendremos un error

## NUESTRO PRIMER SCRIPT

Estamos listos para probar y comentar nuestro primer script. Recuerde guardarlo con el nombre que desee, pero siempre con la extensión .htm. Para ver el resultado, cargue la localización y nombre del archivo en la barra de direcciones del navegador. Si no la recuerda, puede poner C: (o unidad en el que lo guardó) en lugar de la dirección y buscar entre las carpetas que le aparezcan, aquella en la cual lo almacenó.

Ejemplo 2.3

Ejemplo que muestra un mensaje por pantalla.

```
<html>
<head>
<title>Nuestro primer script en JavaScript</title>
  <script type=”text/javascript”>
     alert(“¡Hola Mundo!”);
  </script>
</head>

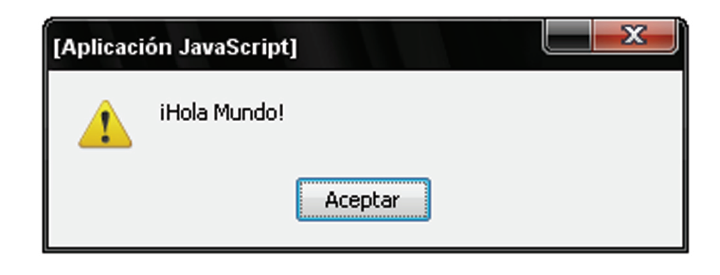
<body>
   <p>Esta página nos sirve para probar nuestro primer script</p>
</body>
</html>
```

Si todo ha salido tal y como debiera, el resultado se parecerá a lo siguiente (con algunas variaciones según el navegador y versión de éste):

Imagen 2.1.

Resultado del ejemplo 2.3.

No es muy original, ya que probablemente se trate del ejemplo más conocido en programación y no haya ni un programador en el mundo que no lo conozca, pero nos sirve para comenzar comentando algunas cosas, observe lo siguiente:



- Hemos colocado el script al comienzo y con las etiquetas <script> correspondientes.
- Hemos utilizado alert para mostrar una pequeña ventana de aviso. Alert en realidad, lo veremos en su momento, es un método perteneciente al objeto window, que se encarga de mostrar mensajes en pantalla.
- Lo que queremos mostrar en pantalla, debemos escribirlo entrecomillado, es lo que llamamos un literal. Los literales siempre se muestran tal y como los escribimos.

## LO QUE HEMOS APRENDIDO: tema 2

→ Qué es un script, una sentencia y una palabra reservada.

→ Cómo escribir las instrucciones de nuestros scripts de forma que JavaScript sea capaz de interpretarlas.

→ A hacer comentarios que facilite la legibilidad de nuestros scripts y su importancia.

→ Hemos explicado en detalle nuestro pequeño script que muestra mensajes en pantalla.

## ✍ ANOTACIONES

# TEMA 3

# Elementos de Programación Básica

- Las variables
- Los tipos de datos
- El alcance de las variables
- Los arrays
- INPUT y prompt
- Operadores: asignación, incremento/decremento, lógicos, aritméticos, etc

**OBJETIVOS:**

- Conocer el que sin duda es uno de los conceptos más importantes de la programación en cualquier lenguaje: las variables.
- Distinguir los distintos tipos de variables que podemos tener y cómo podemos nombrarlas.
- Realizar operaciones con variables utilizando el operador apropiado.
- Saber qué es un array, cómo definirlo.

## LAS VARIABLES

En programación entendemos por variable un espacio que se reservará en memoria y que será utilizado para guardar un dato. El valor de ese dato puede variar a lo largo de la ejecución de nuestro programa, puede ser que necesitemos realizar con él distintas operaciones, consultarlo, etc. Es similar al concepto matemático de variable, aunque si bien algunas de las operaciones que podemos realizar con una variable de programación serían matemáticamente inadmisibles. En contraposición, un valor que no cambia a lo largo de la ejecución de nuestro programa, lo llamaremos constante. En JavaScript no hay una manera especial de declararlas respecto a las variables, e igualmente se reserva espacio para ellas.

Como decíamos, una variable es un espacio reservado para guardar un valor, y a ese espacio le asignamos un nombre. Se trata de una cuestión práctica, pues siempre resultará más fácil recordar el nombre precioVenta que la dirección de memoria 34D0 en hexadecimal (y no digamos ya 0011010011010000 en binario). La cosa se complica además si tenemos en cuenta que es frecuente que un mismo programa utilice cientos de variables.

Nosotros mismos podemos elegir qué nombres dar a nuestras variables. Sin embargo, es aconsejable que usemos nombres significativos, que nos hagan intuir con facilidad qué datos guardamos en ella. Por ejemplo, será mucho más fácil entender a primera vista qué contiene la variable iva8, que la variable llamada solamente con la letra a. El nombre de una variable se puede llamar también identificador.

En cuanto a restricciones al nombre que elijamos debemos tener en cuenta que JavaScript impone las siguientes:

Pueden contener tanto letras como números. Es decir caracteres alfanuméricos.

Pero siempre el primer carácter deberá ser una letra o un guión bajo.

No se admiten espacios en blanco ni otros símbolos como pueden ser: $ % + ( ) / -, etc. Sí, como hemos dicho en el punto anterior, se admite el guión bajo.

Se ha establecido el convencionalismo de utilizar minúsculas en los nombres de variables, excepto en el caso de que se componga de dos palabras, en cuyo caso la primera letra de la segunda palabra se suele escribir en mayúsculas la primera letra. Aunque esto no es estrictamente necesario cumplirlo.

No llevan tildes.

No podemos usar las palabras reservadas para nombrar una variable. (Ver tercer apartado del tema 2)

Siguiendo estas normas, serían válidos los siguientes nombres de variables:

Ejemplo 3.1.

Nombres válidos de variables

```
edad
_irpf
localidad34
codigoPostal
```

A continuación algunos nombres que no serían admitidos, pues no cumplen alguna o varias de las normas expuestas arriba:

Ejemplo 3.2.

Nombres no válidos de variables

```
76 cliente
por%ciento
nombre edad
peso al cuadrado/altura
continue
```

Pongamos un ejemplo sobre cómo podemos usar las variables:

Supongamos que tenemos la variable llamada precio y en ella recogemos los precios de un artículo y otra llamada descuento, que contiene un descuento aplicable a dicho precio (previamente les habríamos asignado dichos valores según veremos en el siguiente apartado). Podemos hacer diferentes operaciones con ellas, por ejemplo:

Ejemplo 3.3.

Expresión de cálculo con variables

```
precioFinal = precio - precio*descuento
```

Obtenemos el precio final del artículo, restando al precio del artículo el correspondiente descuento del precio y todo ello lo guardamos en otra variable llamada precioFinal.

Las variables son uno de los pilares básicos y fundamentales de la programación, de ahí que queramos destacar la importancia de su comprensión. Es prácticamente impensable un programa que haga algo y no utilice al menos una variable. Será por tanto imprescindible que lo entendamos en profundidad desde los inicios. A continuación trataremos los posibles tipos de datos que podemos tener en JavaScript, y por lo tanto asignados a nuestras variables.

## LOS TIPOS DE DATOS

Con ello nos queremos referir a los distintos tipos de valores que por lo general asignaremos tanto a variables como a constantes.

Lo habitual cuando programamos en la mayor parte de lenguajes, y usamos variables, es que como paso previo a la utilización de cada una de ellas, las declaremos. Declarar una variable consiste en “informar” al programa de que vamos a necesitar ese espacio en memoria, cómo lo vamos a llamar y qué tipo de datos guardaremos en él.

En JavaScript esto no es estrictamente necesario. Podemos utilizar una variable sin que exista una declaración previa de ésta. Sin embargo, desde aquí queremos aconsejar que aunque no sea imprescindible, se haga, y se haga correctamente. Adquirir esta costumbre nos puede ser útil a la hora de migrar a otros lenguajes. Recordemos que cuando hablamos de las diferencias entre Java y JavaScript comentamos en su momento que el primero es fuertemente tipado y exige que previa a su utilización, las variables sean declaradas. Si además definimos su tipo, facilitará detectar errores como por ejemplo que pretendamos realizar alguna operación que dicho tipo de dato no admita.

Para declarar una variable tenemos la instrucción var, por ejemplo:

Ejemplo 3.4.

Declaración de una variable.

```
var nombreSocio;
```

Con esta instrucción estamos pidiendo que se nos reserve un espacio en memoria que se llamará nombreSocio. Se pueden declarar varias variables en una sola línea, separando cada una de ellas por medio de comas:

Ejemplo 3.5.

Declaración de tres variables en la misma línea.

```
var nombre, apellidos, direccion;
```

También podemos pedir que nos reserve el espacio y a la vez se le asigne un valor inicial:

Ejemplo 3.6.

Declaración de variable a la vez que se le asigna un valor.

```
var numeroArticulos = 4;
```

En este último caso, numeroArticulos comenzará valiendo inicialmente 4 y se sobreentiende que al ser 4 un número, la variable será numérica. Éste será su valor inicial cada vez que el programa pase por este punto en que se declara. Luego su contenido pueda transformarse y pasar a tener otros valores diferentes, por eso es una variable.

Una vez bien asentado el concepto de variable veamos los tipos de variables que podemos encontrar, o lo que es lo mismo: qué tipos de valores podremos guardar en ellas:

Variables de tipo numérico: pueden almacenar números, tanto enteros (integer) como decimales (float). En el caso de requerir el uso de decimales, hemos de recordar que el formato utilizado es el inglés, y en él los decimales se separan con punto en lugar de coma.

Ejemplos de declaración de variables tipo numérico:

Ejemplo 3.7.

Declaración de variable tipo numérico entero y variable decimal

```
var numeroVecinos = 20;  //ejemplo de variable numérica entera
var precioCoste = 16.04;  //ejemplo de variable numérica decimal
```

En el caso de las variables enteras, puede expresarse también en formato hexadecimal u octal.

Variables de tipo texto: guarda todo tipo de caracteres que serán tratados como texto. Pueden ser palabras, frases, etc. Pueden incluir números, pero entonces carecerán de valor numérico y no podremos realizar con ellos operaciones aritméticas. También se pueden incluir todo tipo de símbolos.

Para distinguirlos, su contenido irá encerrado entre comillas, éstas pueden ser tanto sencillas como dobles. Se les llama cadena de caracteres, en el caso de que dentro se incluya más de uno. También se suele usar la palabra literales para designarlos, ya que su contenido aparecerá literalmente.

Ejemplos de declaración de variables texto:

Ejemplo 3.8.

Declaración de variables asignándoles textos

```
var ciudad = “Tombuctú”;
var unComienzo = ‘Erase una vez en un país muy lejano...’;
var direccionCliente = ”Calle Principal nº 34, bajo dcha.”;
```

En algunas ocasiones nuestras cadenas de texto deben incluir ciertos caracteres que pueden ser problemáticos. Supongamos el caso de que dentro del propio texto debemos incluir comillas. Es el caso del ejemplo que ponemos a continuación, la variable avisoAlumnos será: “los alumnos cuyo apellido comienza por “J” se examinarán mañana”. Esto puede dar lugar a error porque se interpreta como que el texto finaliza después de “por” que es donde se sitúa la segunda comilla, la “J”. En principio, y puesto que se admiten, podemos recurrir a variar el tipo de comillas dentro del texto de la siguiente forma:

Ejemplo 3.9.

Declaración de variable texto que contiene a su vez comillas dentro del texto.

```
var avisoAlumnos = ‘los alumnos cuyo apellido comienza por “J” se examinarán mañana’;
```

Aún así, para evitar posibles errores lo más aconsejable es usar los llamados caracteres de escape. No sólo con las comillas podemos encontrar problemas, existe otra serie de caracteres difíciles de representar, como por ejemplo el INTRO, tabulador, etc. Para todos ellos hay también caracteres de escape. El lector puede encontrar una lista en la correspondiente tabla 3.1 a continuación.. Vemos que todos comienzan con \ ,que es lo que indica que se trata de uno de estos caracteres.

Tabla 3.1.

Lista de los caracteres de escape.

| **Carácter** | **Significado** |
|---|---|
| \n | Nueva línea |
| \t | Tabulador |
| \’ | Comilla simple |
| \” | Comilla doble |
| \\ | Barra invertida |
| \999 | Número ASCII del carácter (p.e. \xA9 muestra el símbolo de copyright). |

Veamos una aplicación práctica en el ejemplo anterior:

Ejemplo 3.10.

Resolución del ejemplo 3.9 utilizando caracteres de escape.

```
var avisoAlumnos = ‘los alumnos cuyo apellido comienza por \“J\” se examinarán mañana’;
```

Observamos en este caso que delante de cada comilla tenemos un \ , esto le indica a JavaScript que no se trata del fin de la cadena de texto, sino de unas comillas que debe representar tal y como aparecen.

Tipo booleano: o tipo lógico, sólo se admiten dos valores: true o false, que corresponden respectivamente a cierto o falso.

Ejemplo 3.11.

Variables de tipo lógico o booleano.

```
pagadaFactura = false; //la factura no está pagada
abonoJoven = true; //cierto que corresponde abono joven.
```

## EL ALCANCE DE LAS VARIABLES

Por alcance o ámbito de una variable entendemos los lugares en los que dicha variable estará disponible, es decir en los que será "visible" y su valor será accesible. Según este criterio las variables pueden ser:

Globales: Cuando una variable es declarada en una página, será legible para toda esa página pero no para otras.

Locales: Son variables declaradas en ámbitos más pequeños, como por ejemplo una función (que más adelante veremos). El contenido o valor de esta variable sólo es accesible dentro de ese pequeño entorno en el cual fue declarada.

Más adelante, cuando veamos las funciones creadas por el programador veremos cómo influye el lugar y el modo en que se declara una variable en cómo es considerada, si local o global.

## LOS ARRAYS

Llamados con frecuencia también vectores, matrices o arreglos, los arrays son un tipo especial de datos que podríamos describir como una colección de variables. Estas variables pueden ser de cualquiera de los tipos vistos anteriormente e incluso entremezclados entre sí.

Pero mejor será que lo veamos con un ejemplo. Supongamos que queremos guardar en variables las asignaturas que se imparten en un centro. Podemos hacerlo según lo visto hasta ahora, de la forma siguiente:

Ejemplo 3.12.

Ejemplo de varias variables relacionadas y cómo declararlas con tipo texto.

```
var asignatura1 = "Matemáticas";
var asignatura2 = "Literatura";
var asignatura3 = "Música";
var asignatura4 = "Física";
var asignatura5 = "Química";
```

Pero sin duda nos resultará mucho más fácil e inteligible de la forma siguiente:

Ejemplo 3.13.

Resolución del ejemplo 3.12 mediante un array o vector.

```
var asignaturas = [“Matemáticas”, “Literatura”, “Música”, “Física”, “Química”];
```

En este caso se trataría de un array. Como vemos los datos que guardamos en él son los mismos, pero la forma es mucho más cómoda. Para hacer referencia a su contenido utilizaremos un índice que indicará la posición del elemento al que queremos acceder, teniendo en cuenta que el primer elemento tiene el número 0. Así por ejemplo, asignaturas[0] será “Matemática”, asignaturas[1] será “Literatura”, etc.

Más adelante veremos que además este tipo de datos cuenta con varias sentencias y operadores especiales. Igualmente veremos que tiene un objeto específico llamado, como era de esperar, array.

## INPUT Y PROMT

Hemos hablado de las variables, su tipo y cómo asignarles un valor. Vamos a hacer un breve inciso para explicar otras dos maneras de hacer llegar a una variable su valor:

Input: Si usted conoce HTML sabrá que es la forma que tiene en los formularios de pedirle al usuario que introduzca un valor al que se le puede asignar un nombre con NAME. Cuando veamos el objeto formulario lo veremos en mayor detalle.

Prompt: Éste le utilizaremos en los siguientes ejercicios cuando necesitemos pedir un valor al usuario, por eso consideramos imprescindible comentarlo ahora. Es muy sencillo de usar y similar a alert, que ya conoce. La sintaxis es como sigue:

```
prompt(texto, valorpordefecto);
```

Donde texto es lo que usted quiera que aparezca en él, de forma literal. Valorpordefecto es el valor que quiere que aparezca por defecto en el cuadro y que será en el caso de que el usuario no lo modifique el que tome la variable. Además le aparecerán dos botones para aceptar o cancelar.

Ejemplo 3.14.

Uso de prompt para pedir un valor al usuario de nuestro programa.

```
<html>
<head>
        <title>Ejemplo prompt</title>

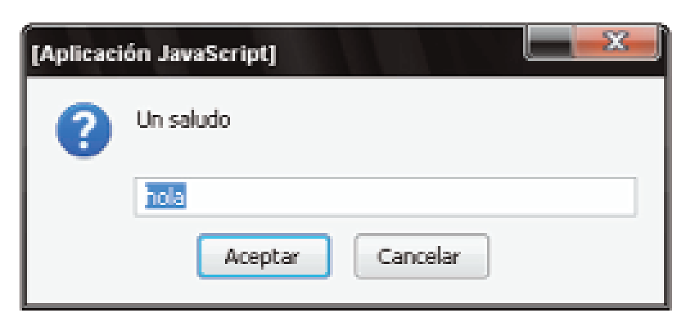
</head>
<body>
<script LANGUAGE=”JavaScript”>
        prompt(“Un saludo”,”hola”);

</script>
</body>
</html>
```

Hará que le aparezca un cuadro más o menos como el siguiente:

Imagen 3.1.

Resultado del ejemplo 3.14.




Si además desea recoger lo que el usuario teclea y utilizarlo, deberá guardarlo en una variable, como en el siguiente ejemplo:

Ejemplo 3.15.

Ejemplo 3.14. guardando lo tecleado por el usuario en una variable para luego usar su valor.

```
<html>
<head>
	<title>Ejemplo prompt</title>

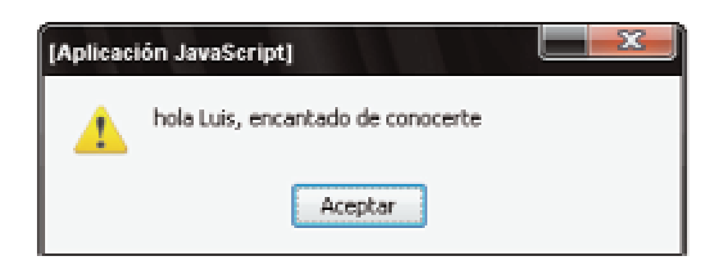
</head>
<body>
	<script LANGUAGE="JavaScript">
	tuNombre=prompt("Un saludo. ¿Cómo te llamas?","");//guardamos lo tecleado en tuNombre
	alert("hola "+tuNombre+", encantado de conocerte");

</script>
</body>
</html>
```

En este caso lo que el usuario teclee en la entrada quedará guardado en la variable tuNombre. Más adelante podemos usar esa variable, por ejemplo para saludar. Si prueba el código anterior y da un nombre (en nuestro caso hemos puesto "Luis"), el resultado será como el que sigue:

Imagen 3.2.

Resultado del ejemplo 3.15.



Observe cómo hemos concatenado varios textos en el mandato alert de forma que compongan el saludo, y cómo si quiere que el espaciado aparezca correctamente deberá incluirlo en el lugar que le corresponda en la cadena. Así, al final de la cadena “hola” hemos incluido un espacio antes de las segundas comillas.

## OPERADORES: ASIGNACIÓN, INCREMENTO/DECREMENTO, LÓGICOS, ARITMÉTICOS, ETC

Una vez vistas las variables y los tipos de éstas, veremos las distintas operaciones que podemos realizar con ellas. Normalmente cuando declaramos una variable lo hacemos para más tarde acceder a su contenido y realizar algún tipo de cálculo con él. Los operadores nos permiten unir identificadores y literales formando expresiones que son evaluadas. JavaScript dispone de muchos operadores, vamos a hacer una clasificación de los más importantes:

- Aritméticos:

Tabla 3.2

Lista de los operadores aritméticos.

| Descripción | Símbolo | Ejemplo | Resultado |
|---|---|---|---|
| Suma | + | 3 + 5 | 8 |
| Resta | – | 45 – 8 | 37 |
| Multiplicación | * | 2 * 14 | 28 |
| División | / | 45 / 5 | 9 |
| Resto división entera | % | 5 % 3 | 2 |

Son muy sencillos y los habituales en los lenguajes de programación y en operaciones normales. Quizás el menos habitual es el resto de la división entera, se trata de eso literalmente. En el ejemplo puesto, si hacemos la división sin decimales, en el resto nos aparecería 2, pues a ese 2 es al que se refiere el resultado.

Incremento o decremento:

Tabla 3.3.

Operadores para incremento o decremento de variable.

| Descripción | Símbolo | Ejemplo | Resultado |
|---|---|---|---|
| Incremento | + + | + + 4 | 5 |
| Decremento | – – | – – 4 | 3 |

Como vemos suma o resta una unidad al valor. Suele ser frecuente su uso en contadores que deban incrementarse o decrementarse en una unidad. Tienen una peculiaridad que vamos a explicar con un ejemplo:

Ejemplo 3.16.

Pre-incremento. La variable b recoge el valor una vez que a se ha incrementado.

```
a = 1;
b = ++a;
```

Comenzamos dando valor 1 a la variable a. La variable b toma el mismo valor de a, pero incrementado en una unidad. Es decir, la variable b valdrá 2. La operación de incrementar en uno se lleva a cabo antes que la operación de la asignación.

Veamos ahora otro ejemplo:

Ejemplo 3.17.

Pos-incremento. En este caso se asigna a b el valor de a, pero se incrementa después.

```
a = 1;
b = a++;
```

Al igual que antes la variable a vale 1. La variable b sin embargo, vale también 1. La razón es que el incremento es posterior a la asignación. Es decir, a b se le asigna primero el valor de a y posteriormente el valor de a se incrementa en una unidad.

De igual forma funciona el operador de decremento, según lo coloquemos antes o después del valor.

- Asignación:

Además del que habitualmente tienen todos los lenguajes de asignación, es decir el igual (a=7, por ejemplo), JavaScript permite combinarlo con otros operadores aritméticos de las formas siguientes:

Tabla 3.4.

Lista de operaciones de asignación y el resultado.

| Operador | Significado |
|---|---|
| a += b | a = a + b |
| a -= b | a = a - b |
| a /= b | a = a / b |
| a *= b | a = a * b |
| a %= b | a = a% b |

- Relacionales o de comparación:

Tabla 3.5.

Lista de las operaciones de relación y resultado de los ejemplos.

| Descripción | Símbolo | Ejemplo | Resultado |
|---|---|---|---|
| Igualdad | = = | 5 = = ‘5’ | CIERTO |
| Igualdad estricta | = = = | 5 = = = ‘5’ | FALSO |
| Desigualdad | != | 5 != 5 | FALSO |
| Desigualdad estricta | != = | 5 != = 5 | FALSO |
| Menor que | < | 5 < 5 | FALSO |
| Menor o igual que | <= | 5 <= 5 | CIERTO |
| Mayor que | > | 5 > 5 | FALSO |
| Mayor o igual que | >= | 5 >=5 | FALSO |

Como vemos los operadores de igualdad y desigualdad estricta hacen una comparación de tipos.

- Lógicos o Booleanos:

Tabla 3.6.

Lista de los operadores booleanos y resultados de los ejemplos.

| Descripción | Símbolo | Ejemplo | Resultado |
|---|---|---|---|
| NO | ! | !(5 = 5) | FALSO |
| Y | && | (5 >2) && (5 <5) | FALSO |
| O | \|\| | (5 = 5) \|\| (5 = 8) | CIERTO |

En el caso del operador Y vemos que se tienen que cumplir las dos condiciones que unimos para que el resultado sea cierto. En el del operador O bastará que se cumpla una de ellas (también pueden cumplirse las dos) para que sea cierto. En los ejemplos puestos, el caso de (5 >2) && (5 <5) se evalúa la expresión, y sólo sería cierto para los valores 3 y 4, como la comparación es con el número 5,obtenemos falso. En el caso de (5 = 5) || (5 = 8) se obtiene cierto, porque al estar unidas con un O y cumplirse la primera condición (5 = 5) es suficiente para resultar cierto.

- Otros operadores:

El menos unario: que vuelve negativo un valor. Por ejemplo -(4+5) dará como resultado – 9.

Concatenación de cadenas: se realiza con el símbolo de la suma. Por ejemplo “la casa” + “ verde” dará como resultado una cadena con la concatenación de ambas, es decir, “la casa verde”.

Operador condicional: asigna un valor u otro en función de que se cumpla o no una condición. Por ejemplo a =3 >2 ? 1 : 3.Le asigna a la variable a el valor 1 dado que se cumple la condición de que tres es mayor que dos.

## LO QUE HEMOS APRENDIDO: tema 3

→ Qué es una variable, cómo nombrarla y darle un valor inicial.

→ Qué valores podemos dar a una variable y cómo operar con ella.

→ Qué es un array y cómo acceder a los distintos elementos que lo componen.

## ANOTACIONES

# TEMA 4

# Estructuras de Control de Flujo

- Estructuras básicas de control de flujo
- Bifurcaciones
- Bucles
- Break y continue
- Anidamiento de dichas estructuras

**OBJETIVOS:**

- Identificar la necesidad que se produce en algunas ocasiones de alterar la secuencia de un programa.
- Aplicar las estructuras de control de flujo a nuestros scripts en aquellas situaciones en que sean necesarias.
- Entender las bifurcaciones condicionales y los bucles.

## ESTRUCTURAS BÁSICAS DE CONTROL DE FLUJO

Es frecuente que cuando programamos tengamos que tomar determinadas decisiones sobre órdenes que serán o no ejecutadas en base a dichas decisiones. Generalmente los programas no tienen una ejecución secuencial, esto es, no suelen ejecutarse sentencia a sentencia en el mismo orden en que fueron escritas, sino que a veces nos resulta necesario alterar este orden previamente establecido. Por ejemplo, puede ser que pidamos al usuario que nos dé una entrada y en función del valor que nos dé, haremos unas cosas u otras. Pensemos en un ejemplo sencillo de un menú en el que el usuario puede escoger la opción que desea ver, deberemos mostrar unas cosas u otras según lo elegido.

Otro ejemplo que podría darse es que tengamos que realizar una misma operación un número de veces seguidas. Supongamos que se nos da el caso de tener que procesar los elementos de un array y realizar una serie de operaciones con todos y cada uno de sus elementos. Sería absurdo que tuviéramos que repetir el mismo código tantas veces como elementos tenga dicho vector, nuestro programa podría llegar a ser demasiado largo. Y desgraciadamente, si el array cambia su dimensión pasando a tener más o menos elementos, el programa dejaría de ser válido, a no ser que lo modificáramos adaptándolo a la nueva situación.

En circunstancias como éstas, podemos utilizar las llamadas estructuras de control de flujo. Dichas estructuras nos permiten alterar la secuencia de instrucciones de forma tal, que no necesariamente se hagan en el mismo orden en el que están escritas. Podríamos decir que nuestros programas serán más “inteligentes”, ya que adquieren una cierta capacidad de tomar decisiones.

Veremos a continuación los tipos de estructuras que tenemos en JavaScript.

## BIFURCACIONES

Son probablemente las estructuras de control más utilizadas en cualquier lenguaje de programación. Capaces de tomar una decisión, de realizar una serie de operaciones, sólo si se cumple una determinada condición. Esta estructura de llama IF, que se traduce como “si” condicional. Establecemos una condición que “si” se cumple, entonces se hará lo que especifiquemos dentro del if.

Su sintaxis es como sigue:

```
if (condicion){
            mandato1...
            mandato2...
            …
            }
```

Pongamos un ejemplo que lo aclare[1]:

[1] *En este caso se utilizan los puntos suspensivos para indicar el resto de las instrucciones del supuesto programa.*

Ejemplo 4.1.

Uso del condicional if.

```
…
if (nota >= 5){
        calificacion = "Apto";
}
…
```

Vemos como la condición que se tiene que cumplir debe ir encerrada entre paréntesis. En este caso, si la nota obtenida es mayor que 5, entonces daremos la calificación de "Apto". Es decir la expresión (nota >= 5) se evalúan y se ve si es o no cierta, sólo en el caso de que ésta sea cierta se realizan las sentencias puesta dentro del if, en el caso que nos ocupa, se da la calificación de "Apto". También observamos que dentro de unas llaves { } deberemos poner todas las instrucciones que se realizarán en el caso de que se cumpla la condición de entrada en el condicional, para nuestro caso sólo tenemos una asignación a una variable (calificación), pero podían ser más sentencias. Se trata como un bloque de instrucciones todas las que estén encerradas entre llaves.

Podría darse la situación de que necesitemos poner varias condiciones para ser evaluadas, en tal caso echaremos mano de los operadores lógicos de la siguiente manera:

Ejemplo 4.2.

Uso de if con dos condiciones unidas con Y.

```
...
If ((total>1000) && (prontoPago == true)){
        descuento = 0.15;
}
...
```

En este caso se deben cumplir dos condiciones: que el total supere la cantidad de 1000 y además que la variable prontoPago sea cierta . Recordemos que al estar unidas por Y, no es suficiente con que se cumpla una de las condiciones, es decir que si el total asciende a más de 1000 euros pero no tiene prontoPago como cierta, la expresión ((total>1000) && (prontoPago == true)) no se evalúa como cierta y por lo tanto no se aplicaría el descuento (ver operadores lógicos o booleanos, tema 3).

En los dos ejemplos anteriores se especificó qué se tiene que hacer cuando se cumple una determinada condición. Pero no se dice nada para el caso contrario, por lo que si se diera, no se haría nada. Supongamos que si una nota es mayor o igual a cinco, damos al alumno por apto. En el caso contrario, es decir que la nota no supere el cinco, determinamos que es "No apto". Entonces:

Ejemplo 4.3.

Uso de if y else

```
…
if (nota >= 5){
          calificacion = “Apto”;
}
else
{
calificacion = “No apto”
}
...
```

En este segundo caso, si la nota no llegara a superar el 5, la variable calificacion pasará a valer “No apto”. Else se interpreta como “en caso contrario”.

Incluso podría darse el caso de que quisiéramos establecer varias operaciones según los valores que tome una variable, es decir, tener varias condiciones:

Ejemplo 4.4.

Uso de if con varios else.

```
...
if(edad < 21) {
 abono=”Joven”;
}
else if(edad < 65) {
 abono=”Normal”;
}
else {
 abono=”Tercera edad”;
}
...
```

Vemos que se irán evaluando una a una las distintas condiciones y se realizan los mandatos correspondientes. Cuando no se cumpla ninguna de ellas, se realizará el último else que aparece al final sin condición de entrada. Como deducimos fácilmente, el mandato else es opcional.

## BUCLES

Entendamos por bucle un conjunto de sentencias que se repite un número determinado de veces. Supongamos que por ejemplo queremos tomar el contenido de un array y sumarle una cantidad. Con la estructura if sólo conseguimos que lo que está dentro de las llaves se ejecute una sola vez, por lo que no sería válida para lo que perseguimos.

Para este tipo de necesidades tenemos las estructuras llamadas bucles. En ellos se establece una condición (igual que lo hacíamos en el if) que se evaluará para ver si se cumpla o no. Ésta será la condición de entrada al bucle. Las instrucciones interiores, encerradas entre llaves, se realizan mientras se siga cumpliendo la condición especificada en la entrada del bucle.

Vamos a ver ahora dos tipos de bucles.

- Bucle FOR: la sintaxis es como sigue:

```
for(var valorInicio; condicion; incremento) {
instrucciones dentro del bucle
}
```

Dentro del paréntesis podemos distinguir las tres partes siguientes:

1. La primera en la cual damos el valor inicial a la/s variable/s que controlarán que el bucle se ejecute.
2. La segunda en la que se establece la condición que se debe cumplir para continuar la ejecución del bucle.
3. La tercera en la que variamos los valores a la/s variable/s de control del bucle.

Quedará mucho más claro si estudiamos un ejemplo sencillo:

Ejemplo 4.5.

Uso de for para realizar un bucle.

```
…
var mensaje1 = “Esta es la vuelta número “, mensaje2 =” de nuestro bucle”;
for(var i = 1; i <=5; i++) {
 alert(mensaje1+ i +mensaje2);
}
...
```

Analicemos en detalle la parte que tenemos dentro del paréntesis en el for: (var i = 1; i < =5; i++) y veamos cada uno de los apartados separados por medio de punto y coma. Son los que determinarán el número de veces que se hará el bucle:

- var i = 1; establecemos que la variable i comenzará tomando un valor de uno, éste será su valor inicial. Este apartado sólo lo tendrá en cuenta la primera vez que se ejecute el bucle, es decir las siguientes veces no se volverá a dar el valor 1 a i. Es habitual darles a estas variables el nombre i de índice, aunque se podría nombrar siguiendo las normas que seguimos con cualquier otra variable, sí que es cierto que se trata de una convención asumida por gran parte de los programadores.
- i <=5; es la condición del bucle. Mientras i tenga un valor inferior o igual a cinco, se realizará lo que haya dentro de él. Así en la primera ejecución, como i es uno, y uno es menor o igual que cinco, se ejecutará.
- i++: se trata de un incremento de la variable i. Si no ponemos este incremento, i valdría siempre uno y por lo tanto nos encontraríamos en un bucle infinito, ya que jamás se llegaría a cumplir la condición de salida: que i

llegue a 5. El bucle se repetiría indefinidamente para i =1. Con i++ conseguimos que en la segunda vuelta i pase a valer dos. Como sigue siendo menor o igual que 5 se vuelve a ejecutar el bucle, mostrando el mensaje. En la siguiente, i se incrementa de nuevo y pasa a valer tres, se vuelve a cumplir la condición. Así se repite hasta que a base de incrementar i llegamos al valor 5, se ejecuta el bucle y a continuación se incrementa i en uno, pasará a valer 6. Como 6 no es menor o igual que 5, el bucle no se realiza y se interrumpe, continuando con el programa en la línea inmediata después del bucle.

- Bucle FOR...IN:
  Se trata de una estructura de control derivada del for que acabamos de ver. Su sintaxis es:

```
for(indice in array) {
 ...instrucciones dentro del for
}
```

Esta estructura nos permite recorrer todos y cada uno de los elementos contenidos en un array. Pongamos un ejemplo que nos lo aclare:

Ejemplo 4. 6.

Uso de for...in para recorrer un array y mostrar sus elementos.

```
var colores = ["Rojo", "Verde", "Azul", "Negro"];
for(i in colores) {
 alert(colores[i]);
}
```

En este caso concreto nuestro contador comienza valiendo 0 y muestra el primer elemento del array. Después 1 y muestra el segundo. Así sucesivamente hasta llegar a 3. El número de elementos irá de 0 a 3, en total 4 elementos, ya que recordaremos que el primer elemento de un array lleva el número 0 (ver tema 3).

Como vemos, esta estructura tiene la peculiaridad de que no necesitamos controlar la variable, ya que la propia estructura lo realiza. No necesitamos iniciar la variable i, ni incrementarla, ni establecer la salida del bucle, cuando llega al final de los elementos, el bucle termina. No necesita saber cuántos elementos componen el vector, es independiente de si el número de ellos aumenta o disminuye. En realidad, nosotros podríamos hacer lo mismo utilizando un bucle for, pero entonces tendríamos que llevar el control del número de ejecuciones nosotros.

## BREAK Y CONTINUE

Lo habitual cuando definimos un bucle es que se ejecute las veces que hayamos previsto, sin más. Pero puede ocurrir por cualquier circunstancia deseemos alterar dicha ejecución. Podemos hacerlo de alguna de las siguientes maneras:

- Llegados a un número de veces o cumpliéndose una condición deseamos que se interrumpa e independientemente de las veces que queden por hacerse, no se lleven a cabo. Para ello usaremos la orden break. Ésta realiza una interrupción abrupta del bucle y continuará con la ejecución de la primera instrucción inmediata al bucle.
- Queremos alterar el orden de forma que se vuelva a repetir el bucle, interrumpiendo la iteración actual. Entonces usaremos continue. Podemos hacer que se "salte" alguna de las ejecuciones de nuestro bucle.

Sin embargo, el hecho de que existan estas sentencias no quiere decir que se deban utilizar indiscriminadamente. Realmente si nuestro bucle está correctamente planteado no deberían ser necesarios. Además digamos que un planteamiento correcto de las estructuras sin necesidad de estas sentencias producen un código más “elegante”. En lo posible las evitaremos.

Veamos un par de ejemplos que nos lo aclaren:

Ejemplo 4.7.

Uso de break para interrumpir un bucle

```
for (i=0;i<10;i++){
   document.write (i)
  escribe = dime si continúo
  if (escribe == “no”) {
    break
  }
}
```

Ejemplo 4.8.

Uso de continue para alterar las vueltas de un bucle.

```
var i=0
while (i<7){
  incrementar = dime si incremento
  if (incrementar == “no”) {
   continue
 }
  i++
}
```

## ANIDAMIENTO DE DICHAS ESTRUCTURAS

También es muy frecuente es que tengamos necesidad de escribir un bucle dentro de otro, o dentro de un condicional. No hay mayor problema, podemos hacerlo, a esto se le llama anidamiento. Lo que sí debemos tener en cuenta es qué sentencias pertenecen a uno y cuáles a otro, ya que una incorrección a la hora de cerrar los distintos bloques, puede generar problemas en la ejecución del programa. Para ayudarnos a hacer un código más claro, se suele recurrir a escribir las sentencias que van juntas tabuladas hacia la derecha:

Ejemplo 4.9.

Anidamiento de varias estructuras unas dentro de otras.

```
var mensaje1 = "Esta es la vuelta número ", mensaje2 =" de nuestro bucle";
for(var i = 1; i <=5; i++) {
  alert(mensaje1+ i +mensaje2);
  if (i=3){
     for(var v = 1;v <=3; v++) {
        alert("esta es la vuelta número tres y voy a decir esto tres veces más");
     }
  }
}
```

En el ejemplo anterior se puede ver claramente, distinguiéndolo por el tabulador, las sentencias que están anidadas dentro de otras. Otra cosa que nos ayuda a distinguir cada bloque, es localizar las correspondientes llaves de cierre, que como vemos están situadas a la misma altura que el comienzo.

## LO QUE HEMOS APRENDIDO: tema 4

→ A utilizar las estructuras de control de forma que nuestros programas sean capaces de “tomar” decisiones basadas en que se den determinadas condiciones.

→ Cómo establecer bifurcaciones en nuestros programas y controlar la ejecución de un bucle .

→ A alterar la secuencia de un bucle mediante los mandatos break y continue.

→ Realizar anidamientos de bucles y condicionales.

## ANOTACIONES

# TEMA 5

# Funciones predefinidas de Javascript

- ¿Qué es una función?
- Funciones que manejan variables de texto
- Funciones que manejan arrays
- Funciones que manejan números

**OBJETIVOS:**

- Conocer cuál es el concepto de función y su utilidad para realizar procesos.
- Saber cómo utilizar las funciones para obtener el resultado buscado.
- Conocer y manejar las funciones predefinidas que nos facilita JavaScript para el manejo de los tipos de datos más importantes.

## ¿QUÉ ES UNA FUNCIÓN?

Es muy frecuente el uso de funciones en cualquier lenguaje de programación. Una función es un conjunto de instrucciones que forman parte de un mismo proceso. Los lenguajes de programación suelen tener muchas funciones predefinidas que realizan los procesos más habituales. Más adelante veremos que nosotros mismos podremos definir las funciones que necesitemos y que el lenguaje no nos facilite. De hecho, alguien se ocupó de programar esas funciones predefinidas para que podamos utilizarlas en nuestro código.

En algunas ocasiones las funciones necesitan valores que debemos enviarles para que ellas puedan procesarlos y devolvernos un resultado. A estos valores que tenemos que enviar a las funciones, los llamamos argumentos (también a veces parámetros) y deben ir entre paréntesis. Aquellas funciones que no necesitan argumentos llevarán los paréntesis vacíos.

También suele ser frecuente que una función retorne el resultado del proceso que realiza. Y puede ser que dicho resultado nos sea útil para continuar la ejecución de nuestro programa. En estos casos lo que haremos es asignar la función a una variable que será la encargada de recogerlo.

Todo esto lo veremos a continuación por medio de ejemplos.

## FUNCIONES QUE MANEJAN VARIABLES DE TEXTO

Son aquellas que se aplican a variables de tipo texto. Veamos una lista de ellas, lo que hacen y algunos ejemplos:

- length(): Calcula el número de caracteres de una variable de tipo texto. Es decir, devuelve el número de caracteres que la componen.

Ejemplo 5.1

Longitud de una variable de tipo texto

```
var miTexto = “Pinté mi casa de amarillo”;
var longitudMiTexto = mensaje.length;
```

El resultado será 25 y se recoge en la variable longitudMiTexto.

- concat(): Realiza la concatenación de dos o más variables de tipo texto. Debemos recordar que el operador + tiene la misma función.

Ejemplo 5.2.

Concatenación de dos variables de tipo texto

```
var primerTexto = “Pinté mi casa”;
var segundoTexto = primerTexto.concat(“ de amarillo”);
```

La variable segundoTexto contendrá la cadena “Pinté mi casa de amarillo”

- toUpperCase(): Transforma todos los caracteres de una variable tipo texto a mayúsculas, independientemente de cómo estuvieran al inicio.

Ejemplo 5.3.

Transformación a mayúsculas de una cadena.

```
var miTexto = "CaSa de DOS pueRTAS";
var textoMayusculas = miTexto.toUpperCase();
```

La variable textoMayusculas contendrá "CASA DE DOS PUERTAS"

Es útil por ejemplo, para realizar posteriores comparaciones de textos, ya que tenemos que tener en cuenta que dos textos escritos en mayúsculas uno y minúsculas otro, no son iguales. En estos casos es más práctico comparar ambos textos convertidos a mayúsculas o en minúsculas, función que se verá a continuación.

- toLowerCase(): Es la función contraria, por así decir, de la anterior. Transforma los caracteres de una variable tipo texto a minúsculas, independientemente de cómo estuvieran al inicio.

Ejemplo 5.4.

Transformación a minúsculas de una cadena.

```
var miTexto = "CaSa de DOS pueRTAS";
var textoMinusculas = miTexto.toLowerCase();
```

La variable textoMinusculas será "casa de dos puertas"

- charAt(posicion): Devuelve el carácter contenido dentro de la variable de texto cuya posición indicaremos dentro de los paréntesis:

Ejemplo 5.5.

Extracción de un carácter de una cadena

```
var saludo = "Muy buenos días";
var letra = saludo.charAt(0); // resultado letra = M
letra = saludo.charAt(4);   // resultado letra = b
```

- indexOf(caracter): Devuelve un número que corresponde a la posición en la que se encuentra el carácter indicado dentro de los paréntesis. Si el carácter se incluye varias veces, indicará sólo su primera posición empezando desde la izquierda. Si el carácter indicado no se encuentra en ninguna posición, la función devuelve el valor -1:

Ejemplo 5.6.

Localización de la posición de un carácter desde el comienzo de la cadena.

```
var saludo = "Muy buenos días";
var posicion = saludo.indexOf('a'); // posicion = 14
posicion = saludo.indexOf('c');   // posicion = -1
```

- lastIndexOf(caracter): Calcula la última posición en la que se encuentra el carácter indicado entre los paréntesis. Si la cadena no contiene el carácter, la función devuelve el valor -1:

Ejemplo 5.7.

Localización de la posición de un carácter desde el final de la cadena

```
var saludo = "Muy buenos días";
var posicion = saludo.lastIndexOf('a'); // posicion = 14
posicion = saludo.lastIndexOf('b');   // posicion = -1
```

Esta función es similar a la anterior y devuelve el mismo resultado, pero comienza a buscar desde el final de la cadena.

- substring(inicio, final): Extrae una parte de una variable de texto, una subcadena de una cadena de caracteres. La primera posición es cero. El segundo parámetro es opcional. Si sólo se indica el primer parámetro, devolverá la parte correspondiente desde dicha posición hasta el final:

Ejemplo 5.8.

Extracción de partes de la cadena, indicando comienzo.

```
var saludo = "Muy buenos días";
var porcion = mensaje.substring(4); // porcion = "buenos días"
porcion = mensaje.substring(6);   // porcion = "enos días"
porcion = mensaje.substring(11);   // porcion = "días"
```

En el caso de que incluyamos las posiciones del inicio y el final, devolverá la parte comprendida entre la posición inicial y la inmediatamente anterior a la posición final:

Ejemplo 5.9.

Extracción de partes de la cadena, indicando comienzo y final.

```
var saludo = "Muy buenos días";
var porcion = saludo.substring(2, 9); // porcion = "y buenos"
porcion = saludo.substring(4, 5);   // porcion = "b"
```

Si indicáramos un inicio negativo, nos devolverá la variable completa:

Ejemplo 5.10.

Extracción de partes de una cadena, indicando un número negativo

```
var saludo = "Muy buenos días";
var porcion = saludo.substring(-2); // porcion = "Muy buenos días"
```

Cuando por error indiquemos un final más pequeño que el inicio, se considerará de forma inversa. El número más pequeño se entenderá que es el de inicio y el más grande el final:

Ejemplo 5.11.

Extracción de partes de una cadena, con los argumentos erróneos

```
var saludo = "Muy buenos días";
var porcion = saludo.substring(5, 0); // porcion = "Muy buenos días"
```

- split(separador): Devuelve un array que contendrá cadenas de texto que serán subcadenas de la variable original, separadas por el carácter separador que hayamos indicado:

Ejemplo 5.12.

Separación de las palabras de una cadena en un array.

```
var mensaje = "Muy buenos días";
var palabras = mensaje.split(" ");// palabras = ["Muy", "buenos", "días"]
```

Permite extraer las letras de una palabra, como en el siguiente ejemplo:

Ejemplo 5.13.

Separación de las letras de una palabra con split.

```
var palabra = "Hola";
var letras = palabra.split(""); // letras = ["H", "o", "l", "a"]
```

## FUNCIONES QUE MANEJAN ARRAYS

A continuación presentamos algunas de las funciones más utilizadas para tratar arrays. Como se verá, algunas son muy similares a las estudiadas para texto:

- length: Devuelve el número de elementos que componen un array.

Ejemplo 5.14.

Obtención del número de elementos de un array.

```
var coloresPrimarios = ["rojo", "azul", "amarillo"];
var numeroColores = coloresPrimarios.length; // numeroColores = 3
```

- concat(): Devuelve la concatenación de varios arrays:

Ejemplo 5.15.

Concatenación de dos arrays en uno.

```
var semana1 = ["lunes", "martes", "miércoles"];
semana2 = semana1.concat("jueves", "viernes");  // semana2 = ["lunes", "martes",
"miércoles", "jueves", "viernes" ]
```

- join(separador): Es la función contraria a split(), vista entre las funciones de texto. Devuelve una variable de texto que es el resultado de unir todos los elementos de un array. Se puede indicar el carácter que deseamos usar de separador.

Ejemplo 5.16.

Unión de todos los elementos de un array en una variable de texto.

```
var array = [“Muy”, “buenos”, “días”];
var saludo = array.join(“”); // saludo = “Muybuenosdías”
saludo2 = array.join(“ “);  // saludo2 = “Muy buenos días”
```

- pop(): Devuelve el último elemento de un array. El array original modifica su longitud disminuyendo en un elemento. (Para quienes conozcan este tipo de estructuras, es como una pila).

Ejemplo 5.17.

Obtención del último elemento de un array

```
var semana= [“lunes”, “martes”, “miércoles”, “jueves”, “viernes”, “sábado”, “domingo”];
var festivo= semana.pop(); // ahora semana= [“lunes”, “martes”,......, “sábado”]
                           // y festivo = “domingo”
```

- push(): Añade un elemento al final del array. El array original se modifica y aumenta su longitud en 1 elemento. (También es posible añadir más de un elemento a la vez).

Ejemplo 5.18.

Añadir un elemento al final de un array

```
var asignaturas= [“Idioma”, “Matemáticas”, “Física” ];
asignaturas.push(“Literatura”);// ahora asignaturas = [“Idioma”, “Matemáticas”, “Física”,
“Literatura”]
```

- shift(): Nos devuelve el primer elemento del array. El array original queda modificado y su longitud disminuye en un elemento. (Para aquellos que conozcan el tipo de estructura, es una cola).

Ejemplo 5.19.

Obtención del primer elemento de un array.

```
var numeros = [1, 2, 3];
var primero = numeros.shift();// ahora numeros = [2, 3], primero = 1
```

- unshift(): Añade un elemento al comienzo de un array. El array original se modifica y aumentará su longitud en un elemento.

Ejemplo 5.20

Añadir un elemento al comienzo de un array

```
var numeros = [1, 2, 3];
numeros.unshift(0);// ahora numeros = [0, 1, 2, 3]
```

- reverse(): Devuelve el array con todos sus elementos colocados en el orden inverso a su posición original:

Ejemplo 5.21.

Ordenación de un array en orden inverso.

```
var array = [1, 2, 3];
array.reverse(); // ahora array = [3, 2, 1]
```

## FUNCIONES QUE MANEJAN NÚMEROS

NaN, viene del inglés, Not a Number, indica un valor numérico no definido (por ejemplo, la división 0/0).

Ejemplo 5.22.

Operación numérica no definida.

```
var numero1 = 0;
var numero2 = 0;
alert(numero1/numero2); // se muestra el valor NaN
```

- IsNaN(): Devuelve si la expresión es o no un número. Permite proteger a la aplicación de posibles valores numéricos no definidos.

Ejemplo 5.23.

Protección ante posibles errores en la división

```
var numero1 = 0;
var numero2 = 0;
if(isNaN(numero1/numero2)) {
 alert("La división no está definida para los números indicados");
}
else {
 alert("La división es igual a => " + numero1/numero2);
}
```

- toFixed(digitos): devuelve el número con tantos decimales como los que indiquemos en el parámetro digitos y realiza los redondeos necesarios.

Ejemplo 5.24

Uso de la función toFixed()

```
var numero1 = 4564.34567;
numero1.toFixed(2); // 4564.35
numero1.toFixed(6); // 4564.345670
numero1.toFixed(); // 4564
```

## LO QUE HEMOS APRENDIDO: tema 5

→ La necesidad de la existencia de funciones para realizar procesos frecuentes.

→ Cómo manejar textos, arrays y números con las funciones predefinidas que nos facilita el lenguaje JavaScript.

## ANOTACIONES

# TEMA 6

# Funciones definidas por el Programador

- Utilidad de las funciones definidas por el programador
- Creación de funciones
- Llamadas a nuestras funciones
- Valor de retorno de una función
- Variables en una función. Ámbito de las variables: globales y locales

**OBJETIVOS:**

- Profundizar en el concepto de función, esta vez creando nosotros nuestras propias funciones.
- Saber cómo definir correctamente nuestras funciones.
- Decidir qué necesidades tendrán nuestras funciones en cuanto a parámetros y variables.

## UTILIDAD DE LAS FUNCIONES DEFINIDAS POR EL PROGRAMADOR

A medida que creamos programas y éstos son cada vez más complejos, suele ocurrir que necesitamos repetir determinadas porciones de código. Supongamos que desarrollamos un script para controlar los pedidos de una pequeña librería. Por cada libro hemos de comprobar que tenemos suficiente cantidad en stock, cual es su precio, finalmente le sumamos el IVA correspondiente y los correspondientes portes en función del lugar de envío.

Podríamos copiar las instrucciones o cálculos necesarios tantas veces como las necesitemos. El problema es que alargaríamos excesivamente nuestro script y en el momento en que necesitemos hacer cualquier modificación la cosa se complicaría, deberíamos buscar entre todas las instrucciones y hacer repetidas veces la misma modificación.

Muchos lenguajes tienen la posibilidad de que el propio programador defina las funciones que pueda necesitar, y utilizarlas cuando necesite, realizando una llamada a éstas de manera similar a las llamadas que realizaría a las funciones predefinidas vistas en el anterior tema.

Además, podrían sernos útiles incluso para otros script, de forma que se convierta en código reutilizable.

Una función es pues, un conjunto de instrucciones o cálculos cuya finalidad es realizar una tarea muy concreta. El propio programador decide el nombre que le asigna y en su caso, los valores que necesitará enviarle para realizar los cálculos. Una función puede además devolver un valor, por ejemplo el resultado de los cálculos que contiene.

Supongamos un ejemplo sencillo en el que necesitáramos sumar dos números, con lo que conocemos hasta el momento, haríamos algo así:

Ejemplo 6.1.

Realización de una suma.

```
//declaración de las variables
var sumatorio;
var sumando1 = 3;
var sumando2 = 5;
//obtención de la suma
sumatorio = sumando1 + sumando2; //sumamos los 2 números
alert("La suma es " + resultado); // mostramos el resultado.
```

No parece muy complicado, pero si más tarde en nuestro programa necesitamos de nuevo sumar otros dos números, volveríamos a poner lo mismo con esos nuevos números:

Ejemplo 6.2.

El mismo del ejemplo 6.1 con otras cantidades.

```
sumando1 = 5;
sumando2 = 8;
sumatorio = sumando1 + sumando2; //sumamos los 2 números
alert("La suma es " + resultado); // mostramos el resultado.
```

Y así cada vez que necesitemos realizar esta operación, con lo cual repetiríamos una y otra vez el mismo código, cambiando las variables a sumar.

Se trata sin duda de un ejemplo simple, pero nos sirve para mostrar la utilidad de las funciones. Podemos crear un apartado con las instrucciones comunes en cada repetición. A dicho apartado le asignaríamos un nombre. Cada vez que necesitemos realizar dichas instrucciones, sólo necesitaríamos hacer una llamada a dicho apartado, por medio del nombre que le pusimos. Ésta sería la idea básica de una función.

Vemos en nuestro ejemplo que siempre se repetirá la suma de los números sumando1 y sumando2, y la instrucción que muestra el resultado. Éstas serán las instrucciones que saquemos a nuestra función.

## CREACIÓN DE FUNCIONES

Comenzaremos creando nuestra función con las instrucciones que hemos determinado que son comunes en cada cálculo. En nuestro caso la suma y muestra del resultado.

Para ello utilizaremos la palabra reservada function y el nombre que decidamos dar a nuestra función. Dicho nombre será el que luego utilicemos para llamarla cuando la necesitemos, por supuesto debe ser único a lo largo de nuestro programa. La sintaxis será la siguiente:

```
function nombre_funcion() {
  instrucciones y cálculos dentro de la función
}
```

En nuestro sencillo ejemplo de suma de dos números, podría ser de la manera siguiente:

Ejemplo 6.3.

Planteamiento de una función que realice lo mismo del ejemplo 6.1 y 6.2.

```
function sumar() {
  sumatorio = sumando1 + sumando2;
  alert("La suma es " + sumatorio);
}
```

Como vemos, dentro de las llaves incluimos todas las instrucciones que necesite nuestra función.

## LLAMADAS A NUESTRAS FUNCIONES

Una vez definida nuestra función ya podemos desde cualquier punto de nuestro programa, hacerle una llamada cuando la necesitemos para sumar dos números:

Ejemplo 6.4.

Llamadas a la función sumar() definida en el ejemplo 6.3.

```
var resultado;
var sumando1 = 3;
var sumando2 = 5;
sumar();
sumando1 = 10;
sumando2 = 7;
sumar();
```

Donde queremos realizar la función, ponemos su nombre seguido de paréntesis. En este caso los paréntesis no llevan nada en su interior, sin embargo en ocasiones puede llevar lo que llamamos argumentos. Los argumentos de una función son aquellos valores que pueden ser necesarios para que la función realice correctamente sus cálculos.

Siguiendo con nuestro ejemplo sencillo, podíamos mejorar la eficacia de nuestra función sumar() si de alguna manera le indicáramos los valores que tiene que sumar y mostrar en el resultado, por ejemplo así:

Ejemplo 6.5.

Modificación de la función del ejemplo 6.4. para que reciba argumentos. Y llamada a ésta.

```
// Definición de la función
function sumar(num1, num2) {
    var sumatorio = num1+num2;
    alert("La suma es " + sumatorio);
}
// Declaración  variables
var sumando1 = 3;
var sumando2 = 5;
// Llamada a la función
sumar(sumando1, sumando2)
```

## VALOR DE RETORNO DE UNA FUNCIÓN

Además de recibir argumentos con los cuales hacer los cálculos o los distintos procesos, una función puede devolver a su vez un valor. Dicho valor puede ser el resultado de las distintas operaciones que realizó la función. Sólo pueden devolver un valor por cada una de las veces que se llama la función.

Para devolver un valor dentro de una función usamos la palabra reservada return. Pondremos return y el nombre de la variable a devolver.

El hecho de que una función tenga esta capacidad de devolver un valor nos permite recoger dicho valor en una variable en el punto en que fue llamada la función y trabajar con ella como lo haríamos con cualquier otra variable.

Ejemplo 6.6.

Función que hace un cálculo y devuelve este valor.

```
function calcularPrecio(precio, impuestos) {
  var portes = 10;
  var precioImpuestos = (1 + impuestos/100) * precio;
  var precioTotal = precioImpuestos + portes;
  return precioTotal;
}
```

Ejemplo 6.7.

Llamada a la función del ejemplo 6.6. y recogida del valor en variables.

```
var precioTotal = calcularTotal(23.34, 16);
var otroPrecioTotal = calcularTotal(15.20, 4);
```

En el caso de que se nos olvidara recoger el valor devuelto por la función JavaScript no produce ningún error, pero dicho valor se perdería

Por último decir que si ponemos instrucciones en una función tras la orden return, éstas serán ignoradas totalmente. Return es una instrucción que obliga a interrumpir la ejecución de la función y retomar el programa principal por la línea siguiente a la que contiene la llamada a la función. Por lo general será pues, la última instrucción de una función.

## VARIABLES EN UNA FUNCIÓN. ÁMBITO DE LAS VARIABLES: GLOBALES Y LOCALES

Entendemos por ámbito de una variable la zona en la cual dicha variable es legible, es decir se puede acceder al valor que contiene.

En JavaScript, al igual que en otros lenguajes, se distinguen dos ámbitos: local y global.

Pongamos un ejemplo que explique todo esto:

Ejemplo 6.8.

Declaración de una variable local en una función

```
function Mensaje() {
  var mensaje = "Saludos";
}
Mensaje();
alert(mensaje);
```

Si ejecutamos este ejemplo veremos que no funciona. Si esperamos que aparezca el contenido de mensaje por pantalla, veremos que esto no ocurre. La razón es que mensaje es una variable local, definida dentro de la función y tendrá validez sólo mientras se esté dentro de dicha función. Cualquier instrucción dentro de la función sí podría usar la variable mensaje,

pero fuera de ella, su valor es inexistente. Si por ejemplo realizamos un pequeño cambio en el código anterior de la forma siguiente:

Ejemplo 6.9.

Declaración y mostrar en la propia función una variable local.

```
function Mensaje() {
  var mensaje = "Saludos";
  alert(mensaje);
}
Mensaje();
```

Veremos que al estar alert dentro de la propia función, sí que nos aparece el mensaje correctamente. En contraposición a las variables locales, tenemos las llamadas variables globales. Éstas sí que son legibles en cualquier punto del programa, incluso dentro de una función.

Ejemplo 6.10.

Variable global.

```
var mensaje = "Saludos";

function muestraMensaje() {
  alert(mensaje);
}
```

En este ejemplo no tenemos ningún problema para mostrar el mensaje dentro de la función, a pesar de que no fue declarada en ella. La función reconoce y muestra correctamente el mensaje, ello es debido a que al estar declarada fuera de la función se considera una variable global y estará disponible en cualquier punto del programa, incluso dentro de una función.

Así pues, cualquier variable declarada fuera de cualquier función se considera automáticamente como variable global, y las declaradas dentro de funciones pueden ser variables globales o locales. Para que una variable declarada en una función sea local, basta con declararla con la palabra reservada var, si no ponemos var se convierte en una variable global.

Ejemplo 6.11.

Declaración dentro de una función de una variable global.

```
function Mensaje() {
  mensaje = "Saludos";
}

Mensaje();
alert(mensaje);
```

Cuando llamamos a una variable local igual que a una global que ya existe, debemos tener en cuenta que las locales tendrán preferencia sobre las globales. Dentro de la función el valor asignado a la variable prevalecerá sobre el valor que tuviera en el resto del programa. Veamos el siguiente ejemplo que lo aclara:

Ejemplo 6.12.

Funcionamiento de las variables locales y globales.

```
var mensaje = “soy global”;

function muestraMensaje() {
  var mensaje = “soy local”;
  alert(mensaje);
}

alert(mensaje);
muestraMensaje();
alert(mensaje);
```

Resultado ejemplo 6.12.

```
Soy global
soy local
soy global
```

Como podemos ver, dentro de la función tendrá el valor que le hemos dado a la variable mensaje, sin embargo cuando finaliza, vuelve a tomar el valor que tenía como variable global.

Si por ejemplo declaramos dentro de una función una variable global (recordemos que esto se hace sin la palabra var) entonces sí modificaría el valor de la variable y al finalizar la función sería el valor que tendría fuera de ella. Lo podemos ver en el ejemplo a continuación:

Ejemplo 6.13.

Funcionamiento variables locales y globales

```
var mensaje = “soy global”;
function muestraMensaje() {
  mensaje = “soy local”;
  alert(mensaje);
}

alert(mensaje);
muestraMensaje();
alert(mensaje);
```

Resultado ejemplo 6.13.

```
Soy global
soy local
soy local
```

En conclusión deberemos estar atentos a cómo declaramos cada variable, ya que su valor puede verse trastocado accidentalmente, y obtener algún resultado inesperado. Es recomendable hacer las variables que sólo se utilizarán dentro de las funciones, y servirán para realizar tareas de dichas funciones, se declaren dentro de la función como variables locales. La variables globales, que se declaren fuera, serán de uso para todo el programa, inclusive para compartir valores con las funciones de una forma sencilla.

## LO QUE HEMOS APRENDIDO: tema 6

→ Crear nuestras propias funciones.

→ A enviar a nuestras funciones los parámetros y establecer las variables que necesitemos dentro de ellas.

→ Determinar el proceso que realizará nuestra función y en su caso, el valor que nos devolverá.

## ANOTACIONES

# TEMA 7

# Otras estructuras de Control

- El bucle while
- El bucle do...while
- Diferencia entre ambos
- Utilizar switch para establecer diferentes condiciones
- Anidamiento

**OBJETIVOS:**

- Ser capaces de plantear estructuras de bucle por medio de dos nuevas sentencias: while y do...while.
- Determinar cuándo aplicamos cada uno de ellos.
- Aprender a usar switch y establecer correctamente cada una de sus condiciones.

## EL BUCLE WHILE

Según la condición de entrada indicada al comienzo del bucle, permite que las instrucciones de su interior se ejecuten varias veces o ninguna.

Su sintaxis es como sigue:

```
while(condicion) {
 …
}
```

While repetirá mientras se cumpla la condición de entrada. En estos casos se hace imprescindible el control de la variable con la cual establecemos la condición del bucle:

Ejemplo 7.1.

Recorre los 30 primeros números, mostrando aquellos que son múltiplos de 3.

```
...
 var numero=30; //variable de control.
  while (numero>0){
   if (numero%3==0){  //si el resto de la división entera es 0, será múltiplo de 3
     alert(numero);
   }
  numero— //disminución en uno de la variable
}
...
```

## EL BUCLE DO...WHILE

Este tipo de bucle es muy similar al anterior. Sin embargo mientras while puede que no se ejecute ninguna vez, do while se ejecutará al menos la primera vez.

Su sintaxis es:

```
do {
 …
} while(condicion);
```

Observamos que la condición se sitúa al final del bucle, es decir se mirará si se cumple una vez ejecutadas las instrucciones de su interior. Es por esto que al menos la primera vez se realizará el bucle, después se observará la condición, si se sigue cumpliendo se volverá a do, en caso contrario no se volverá y proseguirá con la siguiente instrucción inmediata al bucle.

Ejemplo 7.2.

Ejemplo que se realiza al menos una vez a pesar de no cumplirse la condición.

```
...
a=2;
do{
   alert("Saludos");
}while (a>5);
...
```

## DIFERENCIA ENTRE AMBOS

La diferencia entre while y do...while ya la hemos comentado. Consiste en que while puede que, al no darse la condición de entrada en el bucle, no se ejecute ni una sola vez. La razón es que es precisamente la primera instrucción del bucle la que hace la comparación. En la estructura do...while se puede asegurar que al menos se hará una vez. El motivo es que la primera instrucción es do, y no tiene ninguna condición de entrada, luego se hará siempre. La última instrucción while sí que contiene la condición. Entonces, al llegar al final, realiza la comprobación y sólo si la condición es cierta, volverá a la primera instrucción do, realizando por segunda vez el bucle.

## UTILIZAR SWITCH PARA ESTABLECER DIFERENTES CONDICIONES

En el tema 4 vimos también las estructuras de tipo if, condicionales. En el caso de que tuviéramos varias comprobaciones que determinaran las decisiones podíamos utilizar else if.... para especificar cada una de ellas. Cuando dichas decisiones atañen a una única variable el código puede ser redundante:

```
if(numero == 5) {
 ...
}
else if(numero == 8) {
 ...
}
else if(numero == 20) {
 ...
}
else {
 ...
}
```

En casos como estos podemos utilizar la estructura switch que resulta mucho más adecuada, simplificando el código. Es una estructura especialmente pensada para cuando debemos usar condiciones múltiples en base al valor que tome una misma variable.

```
switch(variable) {
 case valor_1:
  …
  break;
 case valor_2:
  …
  break;
 …
 case valor_n:
  …
  break;
 default:
  …
  break;
}
```

Aplicándolo a un ejemplo, tendríamos:

```
…
switch(abono){
case "Joven":
  importe =29.5;
case "Normal"
  importe=40.00
default
  importe=12.5
}
...
```

Como vemos al inicio ponemos switch y la variable con la que vamos a comparar los distintos valores, entre paréntesis, en nuestro caso numero. Las instrucciones que forman parte de la estructura irán entre llaves { }.

A continuación establecemos todos los posibles valores que puede tomar nuestra variable, los posibles casos que se pueden dar de ella. En cada comparación usaremos la palabra reservada case seguida del valor en cuestión. Si nuestro programa llega a este punto y encuentra que el valor coincide, entonces realizará las instrucciones que se especifican dentro del case.

Al final de cada case se suele poner break, aunque no es necesario.

Algo muy importante a tener en cuenta es que siempre se realiza la comparación en el mismo orden en que establezcamos los case.

Si nuestro programa realiza una comparación de la variable y ésta no coincide con ninguno de los valores establecidos en los case, entonces se realizará lo que pongamos en default. Default es opcional, pero si lo incluimos, se ejecutará si tras intentar entrar en los case y no cumplirse la condición.

## ANIDAMIENTO

Lo que comentamos en el tema 4 sobre anidamientos es aplicable igualmente a estas estructuras. También se pueden mezclar entre sí con las del tema 4.

Ejemplo 7.

El ejemplo 4.9. en el que hemos sustituido un bucle for por un bucle while.

```
var mensaje1 = “Esta es la vuelta número “, mensaje2 =” de nuestro bucle”;
i=1;
while (i<=5){
   alert(mensaje1+ i +mensaje2);
   if (i=3){
      for(var v = 1;v <=3; v++) {
         alert(“esta es la vuelta número tres y voy a decir esto tres veces más”);
      }
   }
   i++
}
```

## LO QUE HEMOS APRENDIDO: tema 7

→ Bucles usando while y do...while y qué diferencia cada uno de ellos.

→ A conseguir la ejecución controlada de determinadas sentencias de nuestro programa según el valor que tome una variable usando la sentencia switch.

## ANOTACIONES

# TEMA 8

# DOM

- ¿Qué es DOM?
- Tipos de nodos
- El acceso a los nodos
- La manipulación de los nodos
- Manejo de los atributos de la página mediante DOM

**OBJETIVOS:**

- Conocer qué significa DOM y cómo crea la estructura de las páginas.
- Reconocer los tipos de nodos y la estructura de éstos creada por DOM a partir de una página.
- Aprender cómo manejar el árbol nodos de una página para hacer modificaciones de manera dinámica, como por ejemplo añadir o eliminar elementos/nodos.

## ¿QUÉ ES DOM?

Cada vez que se carga una página el navegador genera de forma automática una estructura que contiene todos y cada uno de los elementos de esa página. A través de mandatos de JavaScript, y otros lenguajes, podemos manipular de forma dinámica estos elementos. El encargado de esto es el llamado DOM.

DOM significa Document Objet Model, y viene a traducirse más o menos como: modelo del objeto documento. Se trata de una de las de las características con mayor influencia en el diseño dinámico de páginas web ya que permite manejar de forma sencilla las páginas. Aunque no haya sido consciente de ello, por el simple hecho de haber realizado alguno de los programas de ejemplo, ya ha utilizado DOM.

Como decíamos, DOM transforma una página en una estructura. Dicha estructura tiene forma de arborescente. Está formado por una serie de elementos llamados nodos, todos ellos conectados entre sí. Todo el contenido de la página estará reflejado en él. Es similar a los árboles genealógicos, de hecho a los nodos que dependen de otros se les suele llamar nodos hijos y al superior, nodo padre.

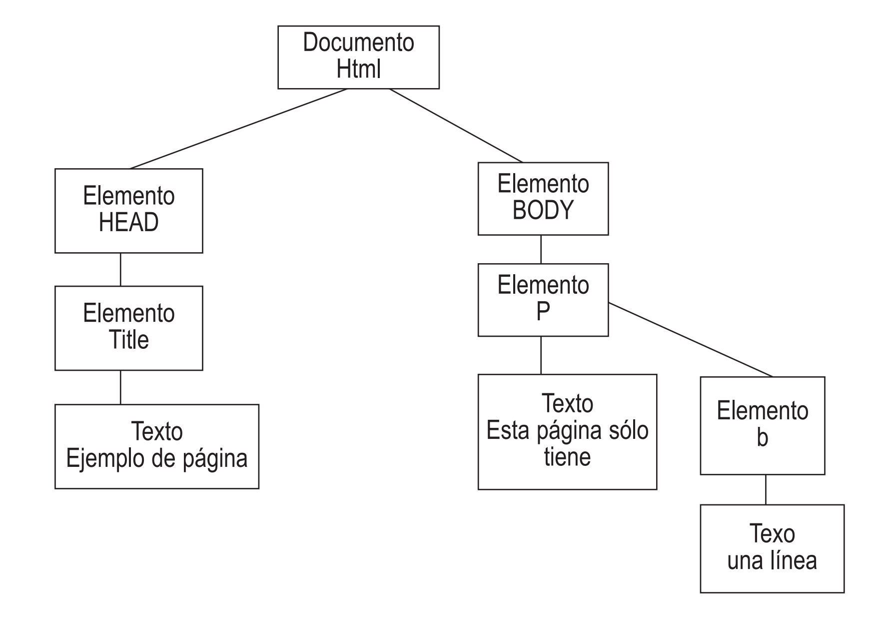
Supongamos una página muy sencilla como la siguiente:

Ejemplo 8.1

Página sencilla.

```
<html>
<head>
<title>Ejemplo de página</title>
</head>
<body>
<p>Esta página sólo tiene <b>una línea</b></p>
</body>
</html>
```

Se transformará en el siguiente árbol:



Este es un ejemplo muy sencillo. Como podemos imaginar, lo habitual en una página cualquiera por la que naveguemos es que tenga muchísimos nodos, cientos de ellos.

## TIPOS DE NODOS

Si bien existen hasta 12 tipos de nodos, para manipular mediante DOM nuestras páginas nos basta con los que detallamos a continuación:

- Documento: es el principal, de él partirán todos los demás nodos que componen nuestra página. También le podemos llamar nodo raíz.
- Elemento: existirá uno por cada una de las etiquetas de HTML. De él pueden derivarse otros nodos.
- Atributo: representa los atributos de HTML.
- Texto: representa el texto de la página.

## EL ACCESO A LOS NODOS

A partir del árbol generado por DOM podemos comenzar a acceder directamente a cada uno de sus nodos y manipularlos para hacer cambios en nuestra página de forma dinámica. Por ejemplo, podríamos querer añadir a nuestro árbol nuevos elementos.

Cada uno de los nodos puede ser accedido de dos formas:

- A través del nodo padre: desde el nodo raíz, indicando toda la sucesión de nodos dependientes de él hasta llegar al nodo en particular
- Mediante acceso directo: que como es de suponer resulta más sencillo y por lo tanto será del que nos ocupemos en este manual.
- Las funciones más utilizadas para acceso directo a los nodos son:
- ElementsByTagName(): se obtiene un array con todos los nodos con una etiqueta HTML determinada de la página.

  Por ejemplo: misParrafos=document.getElementsByTagName("p"); nos daría todas las etiquetas <p>, es decir todos los párrafos de nuestra página.

  Por supuesto podemos acceder a los elementos de dicho array como a los de cualquier otro: misParrafos[3] se refiere al párrafo 4 (recordemos que el primer elemento de un array es el 0).
- getElementsByName(): en este caso se busca el nodo cuyo atributo name (de HTML) sea igual al mencionado. Lo habitual es que sólo exista un elemento en la misma página con ese nombre, por lo que no suele dar problemas. Por ejemplo:

Ejemplo 8.2.

Acceso al nodo "cabecera"

```
var miEncabezado=document.getElementsByName("cabecera");
<p name="cabecera">
…</p>
```

- getElementsById(): se accede por el atributo id de HTML, que debe ser único para cada elemento de la página. Por ejemplo:

Ejemplo 8.3

Acceso al nodo por si id

```
var miPieDePagina=document.getElementsById("pie");
<div id="pie">
...</div>
```

Probablemente sea ésta la forma más habitual de acceso a los nodos, por el id.

## LA MANIPULACIÓN DE LOS NODOS

Podemos manipular los nodos creando alguno nuevo o eliminando alguno existente. Con ello estaremos creando partes nuevas de la página o eliminándolas.

- Para crear un nuevo nodo: supongamos que deseamos añadir un párrafo. Se darán los siguientes pasos:
  - Crear un nodo tipo elemento:
    var miParrafo = document.createElement("p");
  - Crear un nodo tipo texto:
    var miTexto = document.createTextNode("Esto es un texto de ejemplo");
  - Añadir el nodo texto como hijo del nodo elemento:
    miParrafo.appendChild(miTexto);
  - Añadir el nodo elemento como hijo del nodo elemento:
    document.body.appendChild(miParrafo);
- Para borrar un nodo existente: es menos complicado que añadirlo. Supongamos que deseamos borrar el nodo cuyo id es "pie":
  var miPieDePagina=document.getElementsById("pie");
  miPieDePagina.parentNode.removeChild(miPieDePagina);
  Cuando eliminamos un nodo, también eliminamos los hijos que dependan de él, si los tiene.

## MANEJO DE LOS ATRIBUTOS DE LA PÁGINA MEDIANTE DOM

Una vez accedido el nodo, lo habitual es que queramos modificar sus atributos y propiedades. En concreto, podemos cambiar los atributos HTML y CSS de las páginas ya que estos se transforman en propiedades de los nodos.

## LO QUE HEMOS APRENDIDO: tema 8

→ En qué consiste el DOM.

→ La forma de acceder y manejar los nodos del DOM.

## ANOTACIONES

# TEMA 9

# Objetos. El objeto Window

- ¿Qué es un objeto?. Propiedades y métodos
- Objetos predefinidos de JavaScript. Jerarquía de los objetos de JavaScript
- ¿Qué es el objeto window?
- Propiedades del objeto window
- Métodos del objeto window

**OBJETIVOS:**

- Entender el concepto de objeto y qué son sus propiedades y métodos.
- Tener un conocimiento general de los objetos existentes en JavaScript y cuál es la jerarquía por la cual se organizan éstos.
- Conocer el objeto window y su importancia al tratarse del objeto de mayor jerarquía.
- Saber manejar las propiedades y los métodos más importantes del objeto window.

## ¿QUÉ ES UN OBJETO?. PROPIEDADES Y MÉTODOS

Técnicamente un objeto es una materialización de una clase. Se trata de una abstracción por medio de la cual se encapsulan los datos y las rutinas de acceso a éstos. De esta forma se “oculta” la forma en que dicho objeto se hizo y sus detalles. Sólo se facilita la forma de manipular los objetos mediante una interfaz, y se obvia la forma en que fueron creados.

Menos formalmente, un objeto es una estructura que contiene tanto variables, también llamadas propiedades, como funciones que manejan dichas variables, también llamadas métodos.

Basándose en dicha estructura se creó un modelo de programación llamado programación orientada a objetos (POO) que les atribuye ciertas propiedades como son la herencia o el polimorfismo.

Pongamos un ejemplo sencillo de la vida real. Supongamos que usted compra un video doméstico (sería aplicable a cualquier otro aparato). Por lo general usted sabe que alguien diseñó sus circuitos de forma que funcione, pero no le interesa cómo fue diseñado, sino qué puede hacer con su video y de qué forma (propiedades y métodos). Así por ejemplo puede que le interese la forma en la que tiene que programar su video para que grabe con éxito su programa favorito de los miércoles y no la forma en que su video pone en funcionamiento sus circuitos para que se lleve a cabo. Digamos que se realiza una abstracción de cómo funciona internamente su video (objeto), pero se le facilita la forma en que debe manipularlo para que realice lo que usted desea.

En JavaScript los conceptos que generalmente son aplicables a la programación orientada a objetos son algo más difusos y sólo existen objetos.

Para hacer referencia a una propiedad de un objeto la sintaxis que emplearemos es: objeto.propiedad. Para hacer referencia a un método lo haremos de la forma: objeto.metodo(parámetros)

## OBJETOS PREDEFINIDOS DE JAVASCRIPT. JERARQUÍA DE LOS OBJETOS DE JAVASCRIPT

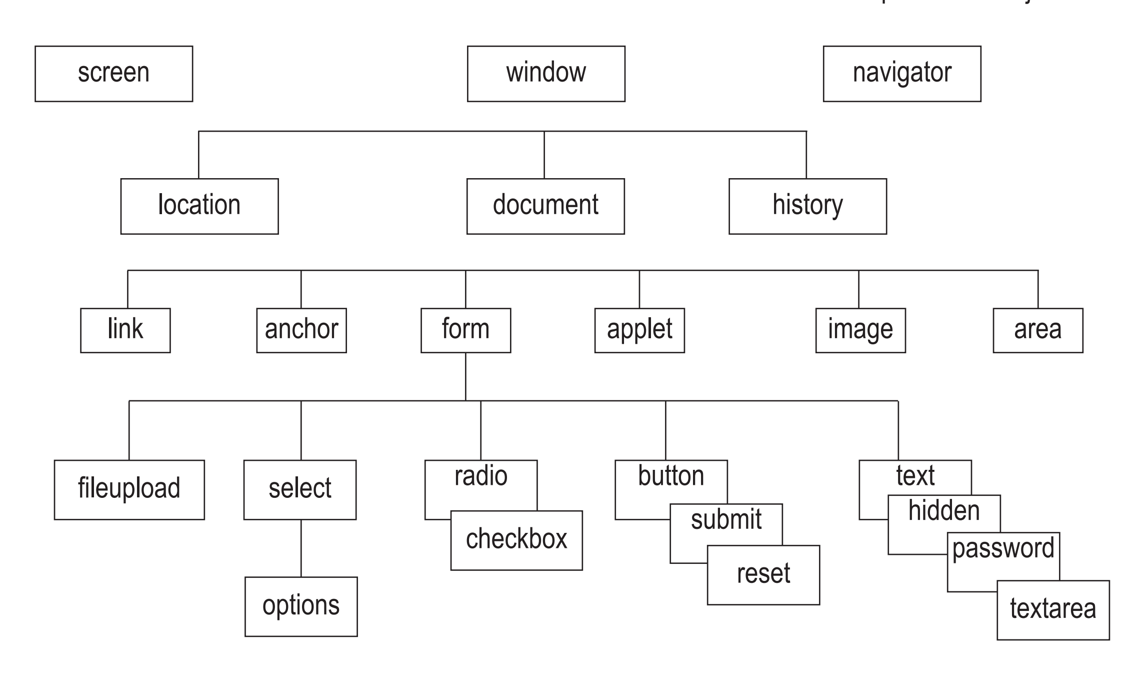
JavaScript dispone de muchos objetos predefinidos que permiten acceder a las funciones normales de un lenguaje. En muchos casos hemos manejado ya objetos (en los ejemplos) aunque no nos hayamos dado cuenta. Es el caso de alert, por ejemplo, que pertenece al objeto window. Es difícil abarcar todos los objetos y todas sus propiedades, pero a lo largo de esta publicación se verán los más habituales e importantes.

El DOM establece una jerarquía entre los objetos como se indica en la imagen (tabla 9.1) a continuación.

Lo habitual en un lenguaje orientado a objetos es que se produjera herencia entre los distintos objetos. En JavaScript esta relación no es de herencia sino de composición, Window se compone de un objeto document, y éste de varios objetos form, image, etc.

Tabla 9.1

Jerarquía de los objetos de JavaScript.




## ¿QUÉ ES EL OBJETO WINDOW?

Como se aprecia en la tabla anterior es el objeto cuya jerarquía es la más alta. Todos los demás objetos posibles de una página web (excepto navigator y screen) están incluidos en él. Con él nos referimos a la ventana actual, la que en ese momento estemos viendo.

## PROPIEDADES DEL OBJETO WINDOW

- closed. Nos dice si la ventana está cerrada, en cuyo caso su valor será true, o abierta, false.
- defaultStatus. Es el texto que aparece en la barra de estado del navegador. Es una cadena.
- frames. Contiene en un array cada uno de los frames de la ventana. Estarán en el mismo orden en el que se definan en el documento, siendo el primero el de índice número 0.
- history. Array que guarda las páginas que hemos visitado, se almacena un historial con ellas.
- length. Guarda el número de frames que contiene la página actual.
- location. Localización de la página. Es lo que se nos muestra en la barra de dirección.
- name. Contiene el nombre de la ventana, o frame actual.
- opener. Se refiere al objeto window que lo abrió usando el método open().
- parent. Se refiere al objeto window que contiene el frameset.
- self. Nombre alternativo del window actual.
- status. Cadena con el mensaje que tiene la barra de estado.

- top. Nombre alternativo de la ventana del nivel superior.
- window. Nombre alternativo del objeto window actual.

## MÉTODOS DEL OBJETO WINDOW

- alert(mensaje). Muestra el mensaje en un cuadro de diálogo
- blur(). Elimina el foco del objeto window actual.
- clearInterval(id). Elimina el intervalo referenciado por ‘id’ (ver el método setInterval(), también del objeto window).
- clearTimeout(nombre). Cancela el intervalo referenciado por ‘nombre’.
- close(). Cierra el objeto window actual.
- confirm(mensaje). Muestra un cuadro de diálogo con el mensaje ‘mensaje’ y dos botones, uno de aceptar y otro de cancelar. Devuelve true si se pulsa aceptar y false si se pulsa cancelar.
- focus(). Captura el foco del ratón sobre el objeto window actual.
- moveBy(x,y). Mueve el objeto window actual el número de pixels especificados por (x,y).
- moveTo(x,y). Mueve el objeto window actual a las coordenadas (x,y).
- open(URL,nombre,caracteristicas). Abre una ventana con el nombre que le especificamos y en ella la URL que ponemos en el primer parámetro. Podemos indicar las características de esta ventana, las posibilidades que tenemos son las siguientes:
    - toolbar. Si deseamos que la ventana tenga barra de herramientas o no. Puede ser yes/no ó 1/0.
    - location. Si deseamos que la ventana tenga campo de localización o no. Puede ser yes/no ó 1/0.
    - directories. Si deseamos que la ventana tenga botones de dirección o no. Puede ser yes/no ó 1/0.
    - status. Si deseamos que la ventana tenga barra de estado o no. Puede ser yes/no ó 1/0.
    - menubar. Si deseamos que la ventana tenga barra de menu o no. Puede ser yes/no ó 1/0.
    - scrollbars. Si deseamos que la ventana tenga barras de desplazamiento o no. Puede ser yes/no ó 1/0.
    - resizable. Si deseamos que la ventana pueda ser cambiada de tamaño o no. Puede ser yes/no ó 1/0.
    - width. Ancho que queremos que tenga la ventana en pixels.
    - Height. Alto que queremos que tenga la ventana en pixels.
    - left. Distancia en pixels desde el lado izquierdo de la pantalla a la que queremos colocar la ventana.
    - Top. Distancia en pixels desde el lado superior de la pantalla a la que queremos colocar la ventana.
- prompt(mensaje,valorInicial). Ya lo explicamos y lo hemos utilizado con frecuencia. Nos muestra un cuadro de diálogo con el mensaje especificado y nos permite introducir un valor. Este valor es guardado en una variable de tipo texto. El segundo parámetro es opcional y permite dar un valor inicial a dicha variable.

- scroll(x,y). Desplaza la ventana a las coordenada x, y.
- scrollBy(x,y). Desplaza la ventana los pixels especificados por (x,y).
- scrollTo(x,y). Desplaza la ventana a las coordenadas (x,y).
- setInterval(expresion,tiempo). Una vez transcurrido el tiempo especificado, se evalúa la expresión.
- setTimeout(expresion,tiempo). Una vez transcurrido el tiempo especificado, se evalúa la expresión.

El ejemplo siguiente abre una pequeña ventana, se trata de un ejemplo muy sencillo, pero resulta muy vistoso:

Ejemplo 9.1

Apertura de una ventana.

```
<html>
<head>
  <title>Ejemplo de abrir ventana</title>
</head>
<body>
<script type=”text/javascript”>
 var opciones=”left=100,top=100,width=250,height=150”;
 miVentana = window.open(“”,””,opciones);
 miVentana.document.write(“Una prueba de abrir ventanas”);
</script>
</body>
</html>
```

## LO QUE HEMOS APRENDIDO: tema 9

→ A definir qué es un objeto y algunos conceptos básicos de la programación orientada a objetos.

→ Hemos adquirido una idea general de los objetos existentes en JavaScript y la forma de relacionarse entre ellos.

→ Distinguir el objeto window, sus propiedades y métodos más importantes.

## ANOTACIONES

# TEMA 10

# Eventos

- Eventos
- Tipos de eventos
- Manejadores de eventos
- Tipos de manejadores de eventos

**OBJETIVOS:**

- Reconocer qué es un evento y cuando se produce.
- Igualmente, reconocer los distintos tipos de eventos que pueden ocurrir ante una acción del usuario.
- Utilizar los manejadores de eventos para responder a la acción realizada por el usuario.

## EVENTOS

Un evento es algo que ocurre. Hasta el momento todos nuestros scripts han seguido un orden secuencial en su ejecución, alterado en alguna ocasión por alguna de las estructuras de flujo que hemos visto. Gracias a ellas conseguimos que se ejecuten sólo algunas porciones de código o que se repita un número de veces.

Sin embargo, queremos ir más allá y permitir que los usuarios de nuestras páginas interaccionen con ellas, de forma que éstas respondan a los distintos eventos que se puedan producir. JavaScript permite dicha interacción ya que utiliza el modelo de programación basada en eventos. De esta manera también conseguimos que nuestras páginas sean mucho más útiles.

Podemos decir que nuestros scripts esperan hasta que ocurre algo que puede ser una pulsación, un movimiento del ratón, etc. Entonces es cuando el script responde adecuadamente a dicho suceso. La respuesta puede ser procesar una información, ejecutar una función que previamente se asoció a dicho evento, etc.

## TIPOS DE EVENTOS

Lamentablemente existen numerosas incompatibilidades entre los diferentes navegadores que se hacen especialmente patentes en el modelo de eventos, llegando en algunos casos a tratarlos de forma diferente e incluso a ignorarlos. Existen varios modelos diferentes para el manejo de eventos según el navegador en que se ejecute.

## MANEJADORES DE EVENTOS

Cada elemento o etiqueta HTML posee un listado de eventos que le pueden suceder en algún momento. Igualmente un mismo evento puede le suceder a varios elementos diferentes.

A continuación, una tabla con los eventos más importantes y a qué elementos son aplicables. Nos servirá para consulta a medida que los necesitemos:

Tabla 10.1

| Evento | Descripción | Elementos para los que se aplica |
|---|---|---|
| onblur | El elemento deja de estar seleccionado | <button>, <input>, <label>, <select>, <textarea>, <body> |
| onchange | Un elemento que se ha modificado deja de estar seleccionado | <input>, <select>, <textarea> |
| onclick | Pinchar y soltar el ratón | Todos los elementos |
| ondblclick | Pinchar dos veces seguidas con el ratón | Todos los elementos |
| onfocus | Seleccionar un elemento | <button>, <input>, <label>, <select>, <textarea>, <body> |
| onkeydown | Pulsar una tecla (sin soltar) | Elementos de formulario y <body> |
| onkeypress | Pulsar una tecla | Elementos de formulario y <body> |
| onkeyup | Soltar una tecla pulsada | Elementos de formulario y <body> |
| onload | La página se ha cargado completamente | <body> |
| onmousedown | Pulsar (sin soltar) un botón del ratón | Todos los elementos |
| onmousemove | Mover el ratón | Todos los elementos |
| onmouseout | El ratón "sale" del elemento (pasa por encima de otro elemento) | Todos los elementos |

| | | |
|---|---|---|
| onmouseover | El ratón pasa por encima del elemento | Todos los elementos |
| onmouseup | Soltar el botón que estaba pulsado en el ratón | Todos los elementos |
| onreset | Borrar los datos del formulario. | <form> |
| onresize | Se ha modificado el tamaño de la ventana del navegador | <body> |
| onselect | Seleccionar un texto | <input>, <textarea> |
| onsubmit | Enviar el formulario | <form> |
| onunload | Se abandona la página (por ejemplo al cerrar el navegador) | <body> |

Podemos observar que cada evento se nombra mediante el prefijo on seguido del nombre de la acción en inglés. Así por ejemplo la acción de pasar el ratón por encima de un elemento es onmouseover.

Hay que tener en cuenta que acciones habituales que un usuario lleva a cabo en las páginas, como por ejemplo enviar un formulario, desencadena una sucesión de eventos como son: onmousedown, onclick, onmouseup y onsubmit.

## TIPOS DE MANEJADORES DE EVENTOS

Pero un evento en sí mismo no tiene mayor importancia a no ser por lo que puede llevar asociado. Por ejemplo puede ser que al ocurrir el evento se ejecute determinado código que le hayamos previamente asociado, de esta forma conseguimos que nuestro script responda a las situaciones que se den en el transcurso de su ejecución.

Estas funciones o códigos asociados a los eventos es lo que llamamos manejadores de eventos o en inglés handlers. Se pueden indicar de varias maneras:

Como atributos de los elementos de HTML: se incluye en el propio atributo. Es sin duda la forma más sencilla de hacerlo. Pongamos un ejemplo en el que al pulsar un botón mostramos un mensaje:

Ejemplo 10.1.

Integración dentro del propio código HTML.

```
<input type="button" value="Pulsar" onclick="alert('Esto es un mensaje que responde al
evento de pulsar el botón');" />
```

Como podemos ver, se incluye el evento y una cadena de texto que contiene todas las instrucciones a ejecutar en el caso de producirse.

Veamos otro ejemplo muy habitual:

Ejemplo 10.2.

Respuesta al evento de cargar la página.

```
<body onload="alert('Se acaba de cargar la página');">
 …
</body>
```

En este caso el mensaje se muestra cuando se ha terminado de cargar la página completamente. Probablemente sea el evento onload uno de los más utilizados ya que las funciones que permiten acceder y manipular los nodos del DOM sólo están disponibles una vez que se haya leído por completo y descargado todo lo que contiene (imágenes, objetos, etc.).

Aunque fácil, es la forma menos aconsejable de utilizar los manejadores ya que puede llegar a complicar excesivamente el código. Sólo es recomendable para casos muy simples.

Como funciones externas de JavaScript: cuando el código es más complejo como puede ser validar un formulario, por ejemplo, es mejor reunir todas estas instrucciones y agruparlas en una función externa. Después, al igual que el caso anterior, haremos una llamada en el elemento HTML a dicha función.

Volviendo al ejemplo del botón que muestra un mensaje al pulsarlo, tendríamos:

Ejemplo 10.3.

Llamada a una función en respuesta a un evento..

```
function mostrarMensaje() {
 alert('Esto es un mensaje que responde al evento de pulsar el botón');
}
<input type="button" value="Pulsar" onclick="mostrarMensaje()" >
```

Sencillamente hemos extraído las acciones en una función externa y le hacemos una llamada tras el evento al que debe responder. La llamada a la función la realizaremos de la forma habitual, incluyendo los parámetros entre paréntesis, si los tuviera.

Semánticos: una buena práctica, en cuanto al diseño de aplicaciones y páginas web, es hacer una separación entre la parte del diseño o presentación (CSS), los contenidos (HTML) y programación (en nuestro caso, JavaScript). De esta forma se simplifican los documentos, se producen unos códigos más "limpios", y por ende más fáciles de mantener y modificar. La alternativa de los manejadores semánticos se basa en propiedades del DOM de los distintos elementos HTML, se les asigna a éstos las funciones externas que actúan de manejadores. Así por ejemplo, nuestro botón que mostraba un mensaje, se haría de la forma siguiente:

Ejemplo 10.4.

Utilizando DOM.

```
 // Función externa
function mostrarMensaje() {
 alert('Esto es un mensaje que responde al evento de pulsar el botón');
}
// Asignar la función externa al elemento
document.getElementById("pinchable").onclick = mostrarMensaje;
// Elemento HTML
<input id="pinchable" type="button" value="Pulsar" >
```

Este ejemplo es muy sencillo y quizás pueda parecer que el método es demasiado complejo para algo tan simple, pero pensemos que lo habitual es que se trate de tareas mucho más complicadas. Además, tiene la ventaja que al no mezclar para nada los códigos de JavaScript con el resto, resulta un programa mucho más limpio y profesional.

## LO QUE HEMOS APRENDIDO: tema 10

→ A intuir en qué situaciones se puede producir un evento.

→ Conocer los eventos más importantes.

→ Las distintas formas que tenemos para manejar un evento que se produce.

## ANOTACIONES

# TEMA 11

# El Objeto Documento

- Definición del objeto documento. Propiedades y métodos del documento
- El objeto image. Propiedades de image
- El objeto anchor. Propiedades de anchor
- El objeto link. Propiedades de link

**OBJETIVOS:**

- Conocer el objeto documento y su importancia como objeto que contiene la totalidad de la página.
- Usar las distintas propiedades y métodos para cambiar nuestra página de manera dinámica.
- Utilizar el objeto image para mostrar diferentes imágenes mediante un scroll.
- Conocer los objetos anchor y link.

## DEFINICIÓN DEL OBJETO DOCUMENTO. PROPIEDADES Y MÉTODOS DEL DOCUMENTO

Este objeto es el que contiene toda la página actual que se esté visualizando en el momento. Incluye a su vez textos, imágenes, etc. Depende del objeto window. Nos permitirá añadir de forma dinámica contenido a nuestra página.

- Propiedades:
    - alinkColor. Es el color de los enlaces activos.
    - anchors. Array con los enlaces del documento.
    - applets. Array con los applets del documento.
    - bgColor. Se refiere al color de fondo del documento.
    - cookie. Cadena con los valores de las cookies del documento.
    - domain. Nombre del dominio del servidor que ha servido el documento.
    - embeds. Array con todos los elementos <EMBED> del documento.
    - fgColor. Propiedad que contiene el color del texto.
    - forms. Array con todos los formularios del documento. Éstos a su vez tienen sus elementos que ya veremos en su momento.
    - images. Array que contiene las imágenes del documento.
    - lastModified. Contiene la fecha de la última modificación del documento.
    - linkColor. Almacena el color de los enlaces.
    - links. Array con los enlaces externos.
    - location. Cadena que contiene la URL del documento.
    - referrer. Cadena con la URL del documento que llamó al actual. Es el enlace a través del cual accedió el usuario.
    - title. Cadena que contiene el título del documento actual.
    - vlinkColor. Color de los enlaces que ya han sido visitados.
- Métodos
    - clear(). Limpiar ventana del documento.
    - open(). Abrir escritura sobre un documento.
    - close(). Cerrar escritura sobre el documento actual.
    - write(). Escribir texto en el documento.
    - writeln(). Escribe texto en el documento añadiendo un salto de línea al final.

El siguiente es un ejemplo muy sencillo que utilizando el método write, nos muestra la última modificación que se ha realizado en el documento (el que contiene a la propia página):

Ejemplo 11.1

Muestra la propiedad de la última modificación del archivo

```
<html>
 <head>
   <title>Ejemplo del objeto documento</title>
  <script type="text/javascript">
    function escribir(){
      document.write("Este documento se modificó por última vez el día:");
      document.write(document.lastModified);
    }
    </script>
 </head>
<body onLoad="escribir()">

</body>
</html>
```

## EL OBJETO IMAGE. PROPIEDADES DE IMAGE

Este objeto representa una imagen. Por medio de él conseguiremos algunos efectos como por ejemplo el conocido rollover, que es el que se produce al cambiar la imagen al pasar el ratón sobre ella. Para que funcione, la imagen deberá ser un enlace. Podemos acceder a las imágenes de un documento utilizando el vector de referencias document.images.

- Propiedades
  - Border. El ancho de los bordes de la imagen en pixels
  - complete. Un valor lógico que será cierto siempre y cuando la imagen haya sido cargada completamente.
  - height. El valor en pixels de la altura de la imagen. Sólo lectura.
  - hspace. Contiene el valor en pixels del espacio horizontal de una imagen respecto al texto que la rodea.
  - lowsrc. Es una imagen alternativa que se carga mientras se recibe la primera.
  - name. El nombre asignado a la imagen. Contiene la ruta asociada a dicha imagen.
  - src. Contiene el archivo de la imagen.
  - vspace. Contiene el valor en pixels del espacio vertical de una imagen respecto al texto que la rodea.
  - Width. El valor en pixels del ancho de la imagen.

El siguiente ejemplo realiza un sencillo rollover entre dos imágenes. Al pasar el ratón sobre la primera, cambia a la segunda y vuelve a la primera cuando el ratón sale de la imagen. Para probarlo necesitará dos imágenes, en nuestro caso las tenemos en dos ficheros llamados "Dibujo1.bmp" y "Dibujo2.bmp".

Ejemplo 11.2.

Muestra dos imágenes intercambiándolas al pasar sobre ellas el ratón.

```
<html>
 <head>
   <title>Ejemplo del objeto imagen</title>
  <script type="text/javascript">//probado, funciona guay
    if (document.images){
            var imagenUno=new Image();
            imagenUno.src="Dibujo1.bmp";
            var imagenDos=new Image();
            imagenDos.src="Dibujo2.bmp";
    }
    function primera(nombreImagen){
      if (document.images){
             document[nombreImagen].src=imagenDos.src;}
    }
    function segunda(nombreImagen){
      if (document.images){
             document[nombreImagen].src=imagenUno.src;}
    }

    </script>
 </head>
<body>
<a href="imagenes2.htm" onMouseOver="primera('prueba');" onMouseOut="segunda('prueba');">
 <img name="prueba" src="Dibujo1.bmp" border=0>
</a>
</body>
</html>
```

## EL OBJETO ANCHOR. PROPIEDADES DE ANCHOR

Se refiere a los enlaces externos contenidos en nuestra página.

- Propiedades
  - target. Cadena que contiene el nombre de la ventana o del frame especificado en el parámetro TARGET
  - hash. Cadena con el nombre del enlace, dentro de la URL

- host. Cadena con el nombre del servidor y número de puerto, dentro de la URL
- hostname. Cadena con el nombre de dominio del servidor (o la dirección IP) dentro de la URL
- href. Cadena que contiene la URL completa
- pathname. Cadena con el camino al recurso, dentro de la URL
- port. Cadena con el número de puerto, dentro de la URL
- protocol. Cadena que contiene el protocolo usado, dentro de la URL
- search. Cadena que contiene la información pasada en una llamada a un script, dentro de la URL.

## OBJETO LINK. PROPIEDADES DE LINK

Se refiere a los enlaces externos contenidos en nuestra página.

- Propiedades
    - target. Cadena que contiene el nombre de la ventana o del frame especificado en el parámetro TARGET
    - hash. Cadena con el nombre del enlace, dentro de la URL
    - host. Cadena con el nombre del servidor y número de puerto, dentro de la URL
    - hostname. Cadena con el nombre de dominio del servidor (o la dirección IP) dentro de la URL
    - href. Cadena que contiene la URL completa
    - pathname. Cadena con el camino al recurso, dentro de la URL
    - port. Cadena con el número de puerto, dentro de la URL
    - protocol. Cadena que contiene el protocolo usado, dentro de la URL
    - search. Cadena que contiene la información pasada en una llamada a un script, dentro de la URL

## LO QUE HEMOS APRENDIDO: tema 11

→ A distinguir qué es el objeto documento y cómo realizar modificaciones en él de forma dinámica

→ Realizar modificaciones sobre las imágenes que aparecen en nuestras páginas y a hacer el conocido efecto rollover que cambia una imagen por otra al situar sobre ella el ratón.

→ Una visión general sobre los objetos link y anchor.

## ANOTACIONES

# TEMA 12

## Objeto de Tipos de Datos

- El objeto string. Propiedades y métodos de string
- El objeto number. Propiedades y métodos de number
- El objeto date. Propiedades y métodos de date
- El objeto math. Propiedades y métodos de math
- El objeto array. Propiedades y métodos de array

**OBJETIVOS:**

- Conocer el objeto string, los tipos de datos a los que se refiere, y sus principales propiedades y métodos.
- Conocer el objeto number, los tipos de datos a los que se refiere, y sus principales propiedades.
- Conocer el objeto date, los tipos de datos a los que se refiere, y cómo manejar fechas utilizando sus principales métodos.
- Conocer el objeto math, los tipos de datos a los que se refiere, sus principales propiedades y cómo realizar operaciones matemáticas con sus métodos.
- Conocer el objeto array, los tipos de datos a los que se refiere, y sus principales propiedades y métodos.

## EL OBJETO STRING. PROPIEDADES Y MÉTODOS DE STRING

El objeto string se refiere a las cadenas de caracteres. Cuando en su momento vimos este tipo de variables, pudimos ver que simplemente bastaba con asignar un valor entre comillas a una variable. Ahora vemos las cadenas de caracteres desde una nueva perspectiva y veremos que JavaScript las considera en realidad objetos. Se pueden crear también con el mandato new. Algunas de las propiedades y métodos que señalamos a continuación, ya se vieron al hablar de estas variables.

- Propiedades
  - length. Entero que nos devuelve la longitud en caracteres de la cadena..
  - prototype. Nos permite asignar nuevas propiedades al objeto String.
- Métodos
  - anchor(nombre). Crea un enlace con el nombre recibido como parámetro. Nombre deberá ir entrecomillado.
  - big(). Muestra la cadena en una letra más grande.
  - blink(). Muestra la cadena con un efecto parpadeante..
  - charAt(número). Muestra el carácter que ocupa la posición dentro de la cadena especificado por número..
  - fixed(). Muestra la cadena de caracteres con una fuente monoespaciada.
  - fontcolor(color). Muestra la cadena en el color especificado que se pondrá entrecomillado. Puede ponerse el color en inglés: “red”, “blue”, “yellow”, etc. o siguiendo el formato habitual de: “#RRGGBB” en hexadecimal.
  - fontsize(tamaño). Muestra la cadena al tamaño indicado. El más pequeño es 1 y el mayor 7.
  - indexOf(cadenaBuscada, indice) Devuelve la posición de la primera ocurrencia de la cadena buscada en la actual. Realiza la búsqueda desde el número especificado por indice. En el caso de omitirse el parámetro inicio, comenzará en el primer carácter. Si la búsqueda no tiene éxito, entonces devuelve el valor -1.
  - italics(). Muestra la cadena en cursiva.
  - lastIndexOf(cadenaBuscada,indice). Devuelve la posición de la última ocurrencia de la cadena buscada dentro de la actual. Se busca desde indice, pero esta búsqueda se realiza hacia atrás. En el caso de omitirse, la búsqueda se realiza desde el último carácter.
  - link(URL). Convierte la cadena en un vínculo asignando al atributo HREF el valor de URL.
  - small(). Muestra la cadena con una fuente pequeña.
  - split(separador). Parte la cadena en un array de caracteres. Si el carácter separador no se encuentra, devuelve un array con un sólo elemento que coincide con la cadena original.
  - strike(). Muestra la cadena de caracteres tachada.
  - sub(). Muestra la cadena con formato de subíndice.
  - substring(primer_Indice,segundo_Indice). Devuelve la subcadena que comienza en la posición ‘primer_Indice + 1’ y que finaliza en la posición ‘segundo_Indice’. Si ‘primer_Indice’ es mayor que ‘segundo_Indice’, empieza

por ‘segundo_Indice + 1’ y termina en ‘primer_Indice’. Si hacemos las cuentas a partir de 0, entonces es la cadena que comienza en ‘primer_Indice’ y termina en ‘segundo_Indice - 1’ (o bien ‘segundo_Indice’ y ‘primer_Indice - 1’ si el primero es mayor que el segundo).

- sup(). Muestra la cadena con formato de superíndice.
- toLowerCase(). Devuelve la cadena en minúsculas.
- toUpperCase(). Devuelve la cadena en minúsculas.

Como ejemplo práctico, el siguiente le muestra todas estas propiedades y métodos de un texto que introduzca el usuario previamente:

Ejemplo 12.1.

Uso de las funciones de string sobre un texto dado por el usuario

```
<html>
<head>
  <title>Ejemplo de manejo de objetos cadena</title>
</head>
<body>
<script LANGUAGE=”JavaScript”>
var miCadena =prompt(“Introduzca un texto: “);
var palabras = new Array(); var palabras = miCadena.split(“ “);
with(document){
write(“La cadena es: “+miCadena+”<BR>”);
write(“El número de caracteres de la cadena es: “+miCadena.length+”<BR>”);
write(“Haciéndola ancla: “+miCadena.anchor(“#”)+”<BR>”);
write(“En grande: “+miCadena.big()+”<BR>”);
write(“Parpadeante: “+miCadena.blink()+”<BR>”);
write(“Caracter 4 es: “+miCadena.charAt(3)+”<BR>”);
write(“Fuente FIXED: “+miCadena.fixed()+”<BR>”);
write(“De color rojo: “+miCadena.fontcolor(“#FF0000”)+”<BR>”);
write(“De color verde: “+miCadena.fontcolor(“green”)+”<BR>”);
write(“Fuente Tamaño 7: “+miCadena.fontsize(7)+”<BR>”);
write(“<BR>En cursiva: “+miCadena.italics()+”<BR>”);
write(“La primera <I>a</I> está, empezando a contar por detrás,”);
write(“ en la posición: “+miCadena.lastIndexOf(“a”)+”<BR>”);
write(“Haciéndola enlace: “+miCadena.link(“#”)+”<BR>”);
write(“En pequeño: “+miCadena.small()+”<BR>”);
write(“Tachada: “+miCadena.strike()+”<BR>”);
write(“Subíndice: “+miCadena.sub()+”<BR>”);
write(“Superíndice: “+miCadena.sup()+”<BR>”);
```

```
write("Minúsculas: "+miCadena.toLowerCase()+"<BR>");
write("Mayúsculas: "+miCadena.toUpperCase()+"<BR>");
write("Subcadena entre los caracteres 4 y 6: ");
write(miCadena.substring(3,6)+"<BR>");
write("Entre los caracteres 6 y 4: "+miCadena.substring(6,3)+"<BR>");
write("Subcadenas resultantes de separar en palabras:</B><BR>");
for(i=0;i<palabras.length;i++) write(palabras[i]+"<BR>");
}
</script>
</body>
</html>
```

## EL OBJETO NUMBER. PROPIEDADES Y MÉTODOS DE NUMBER

Este objeto representa el tipo de dato número. Su valor inicial depende de lo que reciba el constructor. Si recibe un número, se inicia con él. Si no se pasa ningún valor entonces será 0. Cualquier valor no numérico hará que valga NaN (Not a Number). Sólo tiene propiedades y además sólo son de consulta y no pueden modificarse.

- Propiedades
    - MAX_VALUE. Valor máximo que se puede manejar con un tipo numérico
    - MIN_VALUE. Valor mínimo que se puede manejar con un tipo numérico
    - NaN. No es un número
    - NEGATIVE_INFINITY. Representación del valor a partir del cual hay desbordamiento negativo (underflow)
    - POSITIVE_INFINITY. Representación del valor a partir del cual hay desbordamiento positivo (overflow)

## EL OBJETO DATE. PROPIEDADES Y MÉTODOS DE DATE

Por medio de este objeto podemos realizar distintas operaciones con fechas. Debemos tener en cuenta que los meses tienen asignados números entre 0 y 11, es decir 0 es enero, el 1 es febrero y así sucesivamente hasta diciembre que es el mes 11. Igualmente los días de la semana van de 0 a 6, siendo el domingo el 0. Las horas deberán especificarse en formato hh:mm:ss.

El objeto date puede crearse vacío o con un valor. En el caso de crearlo vacío se tomará por defecto la del momento de su creación.

Los formatos para crear fechas pueden ser los siguientes:

var Mi_Fecha = new Date(año, mes);

var Mi_Fecha = new Date(año, mes, día);

var Mi_Fecha = new Date(año, mes, día, horas);

var Mi_Fecha = new Date(año, mes, día, horas, minutos);

var Mi_Fecha = new Date(año, mes, día, horas, minutos, segundos);

- Métodos
  - getDate(). Devuelve un entero que corresponde al día del mes (1-31).
  - getDay(). Devuelve un entero que corresponde al día de la semana (0-6).
  - getHours(). Devuelve un entero que corresponde a la hora (0-23).
  - getMinutes(). Devuelve un entero que corresponde a los minutos (0-59).
  - getMonth(). Devuelve un entero correspondiente al mes del año (0-11)
  - getSeconds(). Devuelve un entero correspondiente a los segundos (0-59).
  - getTime(). Devuelve un entero que corresponde al número de milisegundos transcurridos desde el 1 de enero de 1970 hasta el momento actual.
  - getYear(). Devuelve un entero que corresponde al año actual.
  - setDate(diaMes). Establece el día del mes de date.
  - setDay(diaSemana). Establece el día de la semana de date..
  - setHours(hora). Establece la hora de date.
  - setMinutes(minuto). Establece los minutos de date.
  - setMonth(mes). Establece el mes de date.
  - setSeconds(segundos). Establece los segundos de date
  - setTime(milisegundos). Establece la fecha que dista los milisegundos que le demos del 1 de enero de 1970 en date..
  - setYear(año). Establece el año actual en date.
  - toGMTString(). Devuelve una cadena que usa las convenciones de Internet con la zona horaria

## EL OBJETO MATH. PROPIEDADES Y MÉTODOS DE MATH

Sirve para poder realizar operaciones matemáticas. Sus propiedades son constantes matemáticas que no podemos modificar, sólo consultar.

- Propiedades
  - E. Número 'e', base de los logaritmos naturales (neperianos).
  - LN2. Logaritmo neperiano de 2.
  - LN10. Logaritmo neperiano de 10.

    - LOG2E. Logaritmo en base 2 de e.
    - LOG10E. Logaritmo en base 10 de e.
    - PI. Número PI.
    - SQRT1_2. Raíz cuadrada de 1/2.
    - SQRT2. Raíz cuadrada de 2.
- Métodos
    - abs(numero). Valor absoluto de un número..
    - acos(numero). Función arcocoseno. Devuelve un valor en radianes o NaN. 'numero' debe pertenecer al rango [-1,1], si no devolverá NaN.
    - asin(numero). Función arcoseno. Devuelve un valor en radianes o NaN. 'numero' debe pertenecer al rango [-1,1], si no devolverá NaN.
    - atan(numero). Función arcotangente. Devuelve un valor en radianes o NaN.
    - atan2(x,y). Devuelve el ángulo formado por el vector de coordenadas (x,y) con respecto al eje OX.
    - ceil(numero). Devuelve el entero obtenido de redondear 'numero' por exceso.
    - cos(numero). Devuelve el coseno de 'numero' o NaN.
    - exp(numero). Devuelve el valor de elevar E a un número.
    - floor(numero). Devuelve el entero obtenido de redondear 'numero' por defecto.
    - log(numero). Devuelve el logaritmo neperiano de 'numero'.
    - max(x,y). Devuelve el máximo de 'x' e 'y'.
    - min(x,y). Devuelve el mínimo de 'x' e 'y'.
    - pow(base,exp). Devuelve el valor de base elevado a exp..
    - random(). Devuelve un número aleatorio entre 0 y 1.
    - round(numero). Redondea 'numero' al entero más próximo.
    - sin(numero). Devuelve el seno de 'numero' o NaN.
    - sqrt(numero). Devuelve la raíz cuadrada de número.
    - tan(numero). Devuelve la tangente de 'numero' o NaN.

## EL OBJETO ARRAY. PROPIEDADES Y MÉTODOS DE ARRAY

Permite el manejo de arrays. Su contenido puede ser de cualquiera de los tipos que ya vimos cuando comentamos las variables. Para crear un objeto array utilizaremos su constructor.

Ejemplo 12.2.

Creación de un array.

```
a=new Array(15);
```

Nos crea un array con 15 elementos, del 0 al 14. Su longitud se verá afectada en el momento en que añadamos un elemento nuevo, ampliándose ésta. Cada uno de los elementos contenidos en un array pueden ser accedidos mediante un índice de la forma siguiente: miArray[i]. i será un valor comprendido entre 0 y N-1, siendo N la longitud total del array.

Podemos a la vez que creamos un array, dar los valores iniciales, por ejemplo:

Ejemplo 12.3.

A la vez que lo creamos, asignamosvalores.

```
a=new Array(21,"cadena",true);
```

Como podemos observar los tipos dentro de un mismo array pueden incluso ser de diferentes tipos entre sí.

Teniendo en cuenta que dentro de los paréntesis lo que ponemos son los valores iniciales que asignamos a nuestro array, el ejemplo siguiente:

Ejemplo 12.4.

Asignación valores numéricos.

```
a=new Array(2,3);
```

Crea un array que contiene dos valores numéricos: 2 y 3. En ningún caso creará un array bidimensional o matriz de 2 por 3. Si deseamos crear una matriz de 2 por 3, lo haremos creando un array con las filas que deseemos y luego por cada una de ellas, crearemos un array de tres elementos. Para aclararlo, ver el ejercicio propuesto.

- Propiedades
  - length. Nos indica la longitud del array, esto es, cuántos elementos tiene.
  - prototype. Nos permite asignar nuevas propiedades al objeto String.
- Métodos
  - join(separador). Une los elementos de las cadenas de caracteres de cada elemento de un array en una única cadena.
  - reverse(). Invierte el orden de los elementos del array.
  - sort(). Ordena los elementos del array.

## LO QUE HEMOS APRENDIDO: tema 12

→ Tratar las variables de cadena con el objeto string.

→ Por medio del objeto date mostrar los tipos de datos que contienen fechas.

→ Utilizar el objeto array para manejar vectores.

→ A realizar operaciones matemáticas con el objeto math.

## ANOTACIONES

# TEMA 13

# El Objeto Formulario

- Definición del objeto formulario. Propiedades y métodos del formulario
- El objeto cuadro de texto. Propiedades y métodos de los cuadros de texto
- El objeto checkbox. Propiedades y métodos de checkbox
- El objeto radiobutton. Propiedades y métodos de radiobutton
- El objeto button. Propiedades y métodos de button
- El objeto select. Propiedades y métodos de select

**OBJETIVOS:**

- Conocer el objeto formulario y su importancia como contenedor de otros objetos.
- Reconocer la importancia del formulario como medio de comunicación del usuario con la página.
- Saber utilizar los distintos elementos que podemos tener en el formulario, como son los cuadros de texto, botones, etc.

## DEFINICIÓN DEL OBJETO FORMULARIO. PROPIEDADES Y MÉTODOS DEL FORMULARIO

Se refiere al formulario definido en HTML y contiene todos los elementos de dicho formulario. Todos los formularios de un documento están agrupados en un array.

Para hacer referencia a un formulario de un documento podemos hacerlo de dos maneras: por el nombre que le asignamos al crearlo en HTML (atributo NAME) o por medio de su índice correspondiente en el array de formularios del documento.

Una de las utilidades más frecuentes del trabajo con formularios será la comprobación de los datos introducidos en él por parte del usuario antes de que sean enviados.

- Propiedades
  - Action. Se trata de la acción que se debe realizar cuando se envíe el formulario. Contiene la URL o correo en que se procesarán los datos del formulario. Se corresponde con el parámetro ACTION de HTML.
  - elements. Array que contendrá los elementos que componen el formulario. Asigna a dichos elementos un número en el mismo orden en el que se definen, siendo el primer elemento el 0.
  - encoding. Contiene el tipo de codificación del formulario. Corresponde al parámetro ENCTYPE de HTML.
  - method. Contiene el método con el que se va a recibir/procesar la información del formulario correspondiente al atributo METHOD (GET/POST) de HTML.
- Métodos
  - reset(). Reinicializa todas los campos del formulario. Es el mismo efecto que si pulsamos un botón RESET de HTML.
  - submit(). Envía el formulario. Aunque no se pulse, el efecto es el mismo que si pulsáramos un botón de tipo SUBMIT de HTML.

A continuación veremos los elementos que podemos tener en un formulario y que dependerán jerárquicamente de él.

## EL OBJETO CUADRO DE TEXTO. PROPIEDADES Y MÉTODOS DE LOS CUADROS DE TEXTO

Representan los campos de tipo texto de los formularios. Son los que normalmente se introducen mediante la etiqueta <INPUT type="text"> de HTML. El de tipo password es muy similar sólo que el texto tecleado no aparece, en su lugar se ven unos asteriscos o puntos.

- Propiedades
  - defaultValue. Contiene el valor por defecto que se le ha dado al objeto texto. Equivale al atributo VALUE del INPUT de HTML.
  - name. Cadena cuyo contenido es el valor del parámetro NAME de HTML.
  - value. Cadena cuyo contenido es el texto escrito en el campo.
  - maxlength. Número máximo de caracteres que puede llegar a contener el campo de texto.

- Métodos
  - blur(). Se produce cuando el texto pierde el foco.
  - focus(). Se produce al situar el foco sobre el objeto.
  - select(). Se produce al seleccionar el texto del objeto.

## EL OBJETO CHECKBOX. PROPIEDADES Y MÉTODOS DE CHECKBOX

Este tipo de objetos corresponden a la etiqueta <INPUT type="checkbox">. Son los que nos aparecen con un cuadrado al lado del texto y permite que los seleccionemos. El valor correspondiente de seleccionarlo es true, esto es:cierto. Si no se selecciona entonces es false.

- Propiedades
  - checked. Valor tipo booleano. Nos informa sobre el estado del checkbox, será cierto si está pulsado.
  - defaultChecked. Establece el valor por defecto del checkbox.
  - name. Cadena que contiene el valor del parámetro NAME.
  - value. Cadena que contiene el valor actual del checkbox..
- Métodos
  - Click(). Es como si hiciéramos click sobre el checkbox, cambiando su estado.

## EL OBJETO RADIOBUTTON. PROPIEDADES Y MÉTODOS DE RADIOBUTTON

Son elementos que se muestran en el formulario con un círculo al lado y de los cuales debemos elegir sólo uno de ellos. La etiqueta correspondiente en HTML es <INPUT type="radio". Cuando creamos uno de estos elementos, tendremos un objeto radio por cada uno de los botones

- Propiedades
  - Checked: Indica si está seleccionado o no, es un valor booleano.
  - DefaultChecked : Nos devuelve el estado del objeto por defecto.
  - Value : El valor del campo de radio, asignado por la propiedad value del radio.
  - Length : Es el número de objetos radio que forman parte en el grupo.
- Métodos
  - Click(). Es como si hiciéramos click sobre el checkbox, cambiando su estado.

Veamos el siguiente ejemplo en el que se utiliza este objeto para realizar un test. Como vemos a todas las posibles respuestas se le asigna un nombre en común y como valor "bien" o "mal" según sea o no acertada. Cuando se le da al botón de enviar se evalúa, sencillamente haciendo un recuento de las respuestas acertadas.

Ejemplo 13.1.

Ejercicio tipo test con radiobutton.

```
<html>
 <head>
  <title>Ejemplo de radio</title>
  <script type="text/javascript">
   function nota(examen){
     var resultado=0;
        for(i=0;i<examen.elements.length;i++){
          if ((examen.elements[i].type=="radio") &&
           (examen.elements[i].value=="bien") &&
           (examen.elements[i].checked))
            resultado++;
        }
        alert("Acertaste "+resultado+" de 2 preguntas");
        return false;

   }

   </script>
 </head>
<body>
<form name="miFormulario" onSubmit="nota(this)">
<br>El cielo es de color...
<br><input type="radio" name="respuesta1" value="mal">Rosa
<br><input type="radio" name="respuesta1" value="bien">Azul
<br><input type="radio" name="respuesta1" value="mal">Verde
<br>El animal que nos da leche es...
<br><input type="radio" name="respuesta2" value="mal">La abeja
<br><input type="radio" name="respuesta2" value="mal">La gallina
<br><input type="radio" name="respuesta2" value="bien">La vaca
<br><input type="submit" value="Dime mi nota">
</form>
</body>
</html>
```

## EL OBJETO BUTTON. PROPIEDADES Y MÉTODOS DE BUTTON

Tenemos tres tipos de botones: un botón genérico, ‘button’, que no tiene acción asignada, y dos botones específicos, ‘submit’ y ‘reset’. Estos dos últimos sí que tienen una acción asignada al ser pulsados: el primero envía el formulario y el segundo limpia los valores del formulario.

- Propiedades
    - name. Es una cadena que contiene el valor del parámetro NAME.
    - value. Es una cadena que contiene el valor del parámetro VALUE.
- Métodos
    - click(). Realiza la acción de pulsado del botón.

El ejemplo que mostramos a continuación es un botón que al ser pulsado muestra el ya conocido mensaje de “hola mundo”. Se trata de un botón genérico al que asignamos nosotros la función holamundo().

Ejemplo 13.2.

Botón que al pulsar muestra ventana.

```
<html>
<head>
  <script type=”text/javascript”>
  function holamundo(){
    alert(“Hola mundo!”);
    }
  </script>
</head>
<body>
<form>
  <input type=”button” value=”Púlsame” onClick=”holamundo()”>
</form>
</body>
</html>
```

## EL OBJETO SELECT. PROPIEDADES Y MÉTODOS DE SELECT

Se trata de las listas desplegables que aparecen con un botón con una pequeña flecha a un lado y que al pulsar nos aparecen distintas opciones entre las que podemos elegir una. Se corresponden con la etiqueta <SELECT> de HTML. Accederemos al objeto select por medio del nombre asignado con el atributo NAME.

- Propiedades
    - length. Entero que especifica cuántas opciones tiene la lista. Es igual a la cantidad de etiquetas <OPTION> que pongamos.

- name. Cadena de texto que contiene el valor del parámetro NAME.
- options. Array que contiene cada una de las opciones de la lista. A su vez options tiene las siguientes propiedades:
- defaultSelected. Booleano que indica si la opción está seleccionada por defecto.
- index. Entero con la posición de la opción dentro del array.
- length. Entero con el número de opciones del array.
- options. Cadena con código HTML del array.
- selected. Booleano que indica si la opción está seleccionada o no.
- text. Cadena que contiene el texto mostrado en la lista de una opción concreta.
- value. C adena que contiene el valor del parámetro VALUE de la opción.
- selectedIndex. Entero que nos indica cuál de todas las opciones disponibles se encuentra actualmente seleccionada.

El siguiente ejemplo utiliza un objeto select para realizar un pequeño menú desplegable. Cada valor del objeto es la dirección de una página que se carga en la funcion irA() por medio de la propiedad location de window.

Ejemplo 13.3.

Menú realizado con lista desplegable.

```
<html>
 <head>
   <title>Ejemplo de menú desplegable</title>
  <script type=”text/javascript”>
    function irA(menu){
        window.location.href=menu.options[menu.selectedIndex].value;
    }

    </script>
 </head>
<body>
<form name=”miFormulario”>
 <select name=”menu” size=1 onChange=”irA(this)”>
  <option value=””>Visitar
  <option value=”http://www.yahoo.es”>Yahoo
  <option value=”http://www.google.es”>Google
  <option value=”http://www.20minutos.es”>20 minutos
 </select>
</form>
</body>
</html>
```

## LO QUE HEMOS APRENDIDO: tema 13

→ A conocer y manejar las distintas propiedades y métodos del objeto formulario.

→ Reconocer y tratar los distintos elementos que puede contener objeto formulario a través de sus propios objetos para realizar comprobaciones de los datos introducidos por el usuario antes de que se envíen.

## ANOTACIONES

# TEMA 14

# Otros Objetos

- El objeto navigator. Propiedades y métodos de navigator
- El objeto history. Propiedades y métodos d history
- El objeto location. Propiedades y métodos de location
- El objeto frame. Propiedades y métodos de frame

**OBJETIVOS:**

- Conocer otros objetos que podemos tener con JavaScript como son navigator, location y frame.
- Utilizar el objeto history para recorrer hacia delante o hacia atrás el historial de navegación del usuario.
- Utilizar el objeto frame para controlar los marcos de una página.

## EL OBJETO NAVIGATOR. PROPIEDADES Y MÉTODOS DE NAVIGATOR

Nos da cierta información relativa al navegador que utiliza el usuario de nuestras páginas.

- Propiedades
  - appCodeName. Cadena con el nombre del código del usuario.
  - appName. Cadena con el nombre del cliente.
  - appVersion. Cadena con información sobre la versión del cliente.
  - language. Cadena con información sobre el idioma de la versión del cliente, son dos caracteres.
  - mimeTypes. Array que contiene todos los tipos MIME soportados por el cliente.
  - platform. Cadena con la plataforma sobre la que se está ejecutando el programa cliente.
  - plugins. Array que contiene los plug-ins soportados por el cliente.
  - userAgent. Cadena que contiene la cabecera completa del agente enviada en una petición HTTP. Contiene la información de las propiedades appCodeName y appVersion.
- Métodos
  - javaEnabled(). Devuelve true si el cliente permite la utilización de Java, en caso contrario, devuelve false.

## EL OBJETO HISTORY. PROPIEDADES Y MÉTODOS DE HISTORY

Realiza una lista con los sitios por los que ha navegado el usuario. Nos permite movernos hacia delante o atrás a través de las entradas de la lista. Es el historial de navegación.

- Propiedades
  - Current. Contiene la URL de la entrada actual en el historial.
  - next. Contiene la URL de la siguiente entrada en el historial.
  - length. Nos dice cuántas direcciones se han visitado. Es un número entero.
  - previous. Contiene la URL de la anterior entrada en el historial.
- Métodos
  - back(). Carga la URL del documento anterior.
  - forward(). Carga la URL del documento siguiente.
  - go(posicion). Carga la URL del documento especificado por posicion dentro del historial. Posicion puede ser un entero, en cuyo caso indica la posición relativa del documento dentro del historial; o puede ser una cadena de caracteres, en cuyo caso representa toda o parte de una URL que esté en el historial.

## EL OBJETO LOCATION. PROPIEDADES Y MÉTODOS DE LOCATION

Este objeto contiene la URL actual así como algunos datos de interés respecto a esta URL. Su finalidad principal es, por una parte, modificar el objeto location para cambiar a una nueva URL, y extraer los componentes de dicha URL de forma separada para poder trabajar con ellos de forma individual si es el caso. Recordemos que la sintaxis de una URL era:

```
protocolo://maquina_host[:puerto]/camino_al_recurso.
```

- Propiedades
    - hash. Cadena que contiene el nombre del enlace, dentro de la URL.
    - host. Cadena que contiene el nombre del servidor y el número del puerto, dentro de la URL.
    - hostname. Cadena que contiene el nombre de dominio del servidor (o la dirección IP), dentro de la URL.
    - href. Cadena que contiene la URL completa.
    - pathname. Cadena que contiene el camino al recurso, dentro de la URL.
    - port. Cadena que contiene el número de puerto del servidor, dentro de la URL.
    - protocol. Cadena que contiene el protocolo utilizado (incluyendo los dos puntos), dentro de la URL.
    - search. Cadena que contiene la información pasada en una llamada a un script, dentro de la URL.
- Métodos
    - reload(). Vuelve a cargar la URL especificada en la propiedad href del objeto location.
    - replace(cadenaURL). Reemplaza el historial actual mientras carga la URL especificada en cadenaURL.

## EL OBJETO FRAME. PROPIEDADES Y MÉTODOS DE FRAME

Todos sabemos que la ventana del navegador puede ser dividida en varios frames que contengan cada uno de ellos un documento en el que mostrar contenidos diferentes. Al igual que con las ventanas, cada uno de estos frames puede ser nombrado y referenciado, lo que nos permite cargar documentos en un marco sin que esto afecte al resto.

Realmente cada frame se representa con un objeto window, esto quiere decir que el objeto frame tiene todas las propiedades y métodos del objeto window.

- Propiedades
    - closed. Válida a partir de Netscape 3 en adelante y MSIE 4 en adelante. Es un booleano que nos dice si la ventana está cerrada ( closed = true ) o no ( closed = false ).
    - defaultStatus. Cadena que contiene el texto por defecto que aparece en la barra de estado (status bar) del navegador.
    - frames. Es un array: cada elemento de este array (frames[0], frames[1], ...) es uno de los frames que contiene la ventana. Su orden se asigna según se definen en el documento HTML.

    - history. Se trata de un array que representa las URLS visitadas por la ventana (están almacenadas en su historial).
    - length. Variable que nos indica cuántos frames tiene la ventana actual.
    - location. Cadena con la URL de la barra de dirección.
    - name. Contiene el nombre de la ventana, o del frame actual.
    - opener. Es una referencia al objeto window que lo abrió, si la ventana fue abierta usando el método open() que veremos cuando estudiemos los métodos.
    - parent. Referencia al objeto window que contiene el frameset.
    - self. Es un nombre alternativo del window actual.
    - status. String con el mensaje que tiene la barra de estado.
    - top. Nombre alternativo de la ventana del nivel superior.
    - window. Igual que self: nombre alternativo del objeto window actual.
- Métodos
    - alert(mensaje). Muestra el mensaje 'mensaje' en un cuadro de diálogo
    - blur(). Elimina el foco del objeto window actual.
    - clearInterval(id). Elimina el intervalo referenciado por 'id' (ver el método setInterval(), también del objeto window).
    - clearTimeout(nombre). Cancela el intervalo referenciado por 'nombre' (ver el método setTimeout(), también del objeto window).
    - confirm(mensaje). Muestra un cuadro de diálogo con el mensaje 'mensaje' y dos botones, uno de aceptar y otro de cancelar. Devuelve true si se pulsa aceptar y devuelve false si se pulsa cancelar.
    - focus(). Captura el foco del ratón sobre el objeto window actual.
    - open(URL,nombre,caracteristicas). Abre la URL que le pasemos como primer parámetro en una ventana de nombre 'nombre'. Si esta ventana no existe, abrirá una ventana nueva en la que mostrará el contenido con las características especificadas. Las características que podemos elegir para la ventana que queramos abrir son las siguientes:
    - prompt(mensaje,respuesta_por_defecto). Muestra un cuadro de diálogo que contiene una caja de texto en la cual podremos escribir una respuesta a lo que nos pregunte en 'mensaje'. El parámetro 'respuesta_por_defecto' es opcional, y mostrará la respuesta por defecto indicada al abrirse el cuadro de diálogo. El método retorna una cadena de caracteres con la respuesta introducida.
    - scroll(x,y). Desplaza el objeto window actual a las coordenadas especificadas por (x,y).
    - scrollBy(x,y). Desplaza el objeto window actual el número de pixels especificado por (x,y).
    - scrollTo(x,y). Desplaza el objeto window actual a las coordenadas especificadas por (x,y).

- setInterval(expresion,tiempo). Evalúa la expresión especificada después de que hayan pasado el número de milisegundos especificados en tiempo. Devuelve un valor que puede ser usado como identificativo por clearInterval().

- setTimeout(expresion,tiempo). Evalúa la expresión especificada después de que hayan pasado el número de milisegundos especificados en tiempo. Devuelve un valor que puede ser usado como identificativo por clearTimeout().

## LO QUE HEMOS APRENDIDO: tema 14

→ Utilizar el objeto navigator para presentar las características del navegador del usuario.

→ Facilitar al usuario la navegación en nuestras páginas, hacia delante y atrás en las páginas visitadas.

→ Utilización del objeto frame para manejar los marcos de nuestras páginas.

## ANOTACIONES


9788468113784